Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Zacatenco

Departamento de Computación

Multiplicadores de arquitectura segmentada y su aplicación al cómputo de emparejamientos bilineales

Tesis que presenta

Nidia Asunción Cortez Duarte

para obtener el Grado de

Maestra en Ciencias en Computación

Director de la Tesis

Dr. Francisco Rodríguez Henríquez

México, D.F. **Agosto 2009**

II

Resumen

Las aplicaciones criptográficas requieren de implementaciones eficientes de aritmética básica sobre campos finitos tales como suma, resta, multiplicación, división, exponenciación y cálculo de raíces, por mencionar algunas, siendo la multiplicación, la operación que requiere mayor atención debido a su costo en recursos y tiempo de ejecución, además de que es la operación más utilizada en la mayoría de los algoritmos criptográficos.

En esta tesis centramos nuestro interés en los algoritmos de multiplicación; partiendo del algoritmo original de Karatsuba-Ofman, analizamos algunas de sus variantes y describimos las estrategias que consideramos para diseñar una arquitectura la cual esta basada en una estructura segmentada de k-estados. Implementamos nuestras arquitecturas en dispositivos de hardware reconfigurable en el campo finito binario $\mathbb{F}_{2^{239}}$ con $k = 1, \ldots, 3$, donde cada estado está compuesto por un multiplicador de 120-bits. Asimismo, se diseñó un multiplicador para el campo finito ternario $\mathbb{F}_{3^{97}}$ con $k = 1, \ldots, 4$, donde cada estado esta compuesto por un multiplicador de 49-trits. Nuestras arquitecturas son capaces de cálcular en promedio k multiplicaciones cada 3 ciclos de reloj. En el campo $\mathbb{F}_{3^{97}}$ mediante una arquitectura segmentada de 4 estados fue posible calcular más de una multiplicación polinomial por cada ciclo de reloj.

Debido a que una de las aplicaciones más relevantes para este tipo de multiplicaciones es el cálculo de emparejamientos bilineales decidimos estudiarlos en esta tesis. Un emparejamiento necesita el cómputo eficiente de un ciclo principal junto con una operación adicional denominada exponenciación final. En ambos módulos se requiere evaluar muchas multiplicaciones. En este trabajo se decidió utilizar como caso de estudio el diseño del módulo de exponenciación final en característica dos.

Todos los diseños se implementaron utilizando el lenguaje de descripción de hardware VHDL y se sintetizaron para dispositivos Virtex II-Pro y Virtex 5 de Xilinx. Se ocuparon las herramientas de Xilinx para descripción y simulación. De acuerdo a nuestros resultados de la simulación *post-place and route* en las FPGAs de Xilinx, nuestros multiplicadores son los más eficientes y nuestra implementación de la exponenciación final resulta ser competitiva con respecto a otros diseños reportados.

Abstract

Cryptographic applications require efficient implementation of the basic arithmetic finite field operations such as field addition, subtraction, multiplication, division, exponentiation, squaring, square root, cubing and cube root computation. Among the aforementioned arithmetic operations, multiplication is by far the one that has received more attention, mostly because of its crucial role in the aforementioned cryptographic schemes besides the cost of area and time calculation.

In this thesis, we focus our attention in parallel subquadratic multiplication algorithms. From the original Karatsuba-Ofman algorithm, we studied some variants recently published and we also describe the strategies that were considered to design our reconfigurable hardware architecture. We decided to use a k-stage pipeline structure. We implemented our architectures in reconfigurable hardware in the binary finite field $\mathbb{F}_{2^{239}}$ with $k = 1, \ldots, 3$, where each stage is composed of a 120 bit polynomial multiplier unit and over the ternary finite field $\mathbb{F}_{3^{97}}$ with $k = 1, \ldots, 4$, where each stage is composed of a 49 trit polynomial multiplier unit. Our architectures can compute in average k field multiplications every three clock cycles. In the field $\mathbb{F}_{3^{97}}$, our four-stage pipeline design can compute in average more than one field multiplication per clock cycle.

Since the most relevant applications of this kind of multipliers lie on bilinear pairing computation, where several field multiplications need to be computed at once, we decided to study it. A bilinear pairing needs the efficient computation of a main cycle and one additional operation called final exponentiation which is required to obtain a unique value. Both modules imply the computation of many multiplications. Therefore, the final exponentiation was considered as a good case of study for our multipliers in characteristic two.

All the designs were implemented utilizing the hardware description language VHDL and were also sintetized for Virtex II-Pro and Virtex 5 FPGA devices. The Xilinx tools were utilized for description and simulation. According to our post-place and route results on Xilins FPGAs, our multipliers are the most efficient and our design of final exponentiation is competitive compared to previously published works.

Agradecimientos

A mi madre Lidia
Por su apoyo incondicional, por su atención, mi educación y su comprensión. Gracias por ser mi ejemplo de vida.

A mi padre Jesús
Por el amor que día a día me ha demostrado. Gracias por enseñarme distintas maneras de vivir la vida.

A mi hermano Daniel
Por sus palabras de aliento y su constante apoyo.

A Alejandro González
Por escucharme en todo momento, principalmente por su cariño y motivación en la étapa final de esta tesis.

A toda mi familia y amigos
Por sus buenos deseos y su agradable compañia.

A mis amigos
Daniel, Luis, Jésus y Lulú
por sus valiosos consejos y enseñanzas así como su paciencia en aquellas jornadas de trabajo arduo.

Miriam, Adriana
por los grandes momentos que compartimos y su infinita confianza.

Gaby y Amilcar
por toda su ayuda y comprensión.

A Raymundo y Joselyn
Por brindar una opción para darle un respiro a la vida así como sus palabras de aliento.

A las señoras
Flor Córdova, Sofía Reza y Felipa Rosas
por su apoyo, cariño y su infinita paciencia.

A mi director de tesis
El Dr. Francisco Rodríguez Henríquez por
el apoyo brindado durante el desarrollo de esta tesis.

A los doctores
Arturo Díaz Pérez y Luis Gerardo de la Fraga
por sus valiosos comentarios en la revisión y corrección
para el enriquecimiento de este documento.

Al Dr. Jean Luc Beuchat
Un agradecimiento especial por sus consejos que fueron de
suma importancia para obtener los resultados de esta tesis.

CINVESTAV-IPN
Por permitirme ser parte de esta institución por brindarme
la formación y las herramientas suficientes durante mis
estudios de maestría.

CONACYT
Por el apoyo económico que me fue otorgado a partir de su
programa de becas y del proyecto número 60240 para
llevar realizar mis estudios de maestría.

COMECYT
Por el apoyo económico que me fue otorgado a partir de su
programa de becas para posgrado para así llevar a buen
término mis estudios de maestría.

Índice general

Índice de figuras

Índice de cuadros

Introducción

En la actualidad para realizar el intercambio de cualquier tipo de información se debe dec contar con mecanismos o servicios de seguridad. Se requiere de herramientas criptográficas para poder construir dichos servicios, en general, las aplicaciones criptográficas requieren de implementaciones eficientes de la aritmética básica sobre campos finitos tales como suma, resta, multiplicación, división, entre otras. Siendo el tema principal para esta tesis la multiplicación ya que es la operación que consume más recursos y tiempo de operación.

Analizamos el algoritmo de Karatsuba-Ofman y algunas de sus variantes asi como las posibles arquitecturas a implementar como serial, paralela y segmentada.

Una de las aplicaciones más relevantes para este tipo de multiplicaciones es el cálculo de emparejamientos bilineales sobre curvas elípticas definidas en campos finitos. Decidimos tomar como caso de estudio una operación particular en el cálculo de emparejamientos bilineales.

El resto de la tesis se encuentra organizada de la siguiente manera: en el capítulo 1 se presentan las nociones básicas de seguridad brindando una introducción a los conceptos básicos de criptografía. En el capítulo dos se describen los preliminares matemáticos con una descripción general de grupos así como la aritmética en campos finitos binarios y ternarios, se detalla la operación de multiplicación ya que es el principal estudio de esta tesis. En el capítulo 3 explicamos el diseño y la implementación de nuestros multiplicadores en ambas características además incluimos una tabla comparativa de nuestro trabajo con respecto a otros publicados en la literatura abierta de tal manera que nuestros diseños resultan más eficientes que cualquier otro. Como caso de estudio de nuestros multiplicadores en el capítulo 4 detallamos nuestra arquitectura para el cómputo de la exponenciación final de los emparejamientos bilineales. Finalmente en el capítulo 5 presentamos nuestras conclusiones y trabajo futuro.

Capítulo 1
Nociones básicas de seguridad

La comunicación entre dos entidades generalmente se realiza a través de un canal inseguro, lo cual implica que la información intercambiada no cuenta con ninguna garantía de privacidad o integridad. Por lo tanto, el intercambio de información debe ser provisto por mecanismos o servicios que brinden seguridad. Estos mecanismos o servicios de seguridad son construidos mediante herramientas criptográficas.

En este capítulo se presenta una descripción introductoria de los conceptos relacionados con seguridad y criptografía. Se explican de manera breve técnicas criptográficas que ayudarán a entender las bases de este trabajo.

1.1. Introducción a la seguridad y criptografía

En la actualidad, la necesidad de proteger la información es enorme, la mayoría de los problemas de seguridad son causados intencionalmente por gente maliciosa que intenta ganar algo o dañar a alguien.

Los temas de seguridad pueden dividirse en términos generales en cinco áreas interrelacionadas [51]:

Confidencialidad: protege la información ante revelaciones no autorizadas.

Autenticación: verifica que un nodo o un usuario sea quien dice ser.

Integridad: protege los datos del sistema ante modificaciones o alteraciones no deseadas.

No repudio: impide que un emisor niegue haber enviado un mensaje o que un receptor niegue haberlo recibido.

Control de acceso: protege los recursos del sistema.

Los servicios de seguridad se ofrecen a través de herramientas criptográficas, recurriendo a protocolos de seguridad o implementando algoritmos criptográficos directamente [42]. Usualmente, los servicios de seguridad son elaborados en base a esquemas o primitivas de seguridad más elementales, de esta forma su arquitectura puede ser visualizada mediante un modelo de capas. La Figura 1.1 muestra el esquema jerárquico de seis capas para un

sistema de seguridad.

Analizando dicha figura de arriba hacia abajo, se observa que en la capa 6 se listan diversas aplicaciones comunes de seguridad tales como: correo electrónico seguro, monedero digital, elecciones electrónicas, etc., todas éstas dependen de la implementación, en la capa 5, de los protocolos de comunicación como SSL/TSL, IPSec, IEEE802.11, etc. Sin embargo, todos estos protocolos no pueden funcionar sin haberse implementado la capa 4, que trata los servicios de seguridad descritos previamente, es decir, autenticación, integridad, no repudio y control de acceso. La infraestructura fundamental para tales servicios de seguridad se basa en dos pares de primitivas criptográficas descritas en la capa 3, llamadas, cifrado/descifrado y firma digital/verificación. Todas estas primitivas pueden ser implementadas por la combinación de los algoritmos de llave pública y privada, los cuales se listan en la capa 2. Ahora bien, con el fin de obtener un alto desempeño de los algoritmos criptográficos es indispensable contar con implementaciones eficientes de las operaciones aritméticas básicas que se muestran en la capa 1.

Figura 1.1: Modelo jerárquico para aplicaciones

El interés de esta tesis se centra en las capas 1 y 2. Con la finalidad de tener un panorama general de las bases que sustentan este trabajo, a continuación se presentan algunos conceptos importantes.

1.2. Criptografía

La palabra criptografía proviene del griego ***kryptos***, que significa esconder y ***graphein***, escribir, es decir, escritura escondida. La criptografía ha sido usada a través de los años para mandar mensajes confidenciales que sólo las personas autorizadas pueden entender o descifrar. Una forma muy simple de definirla es como: **"la ciencia de ocultar mensajes de forma segura "** [46].

El proceso de ocultamiento inicia con la transformación del texto legible original denominado *texto claro* o *texto plano* en un *texto cifrado* ilegible. La transformación se realiza mediante una función E parametrizada por una llave k_1. Por otra parte, el proceso inverso se refiere a la obtención del *texto plano* a partir del *texto cifrado* con la ayuda de una función D parametrizada por una llave k_2.

Las llaves usadas en ambos procesos (cifrado y descifrado respectivamente) pueden ser iguales o distintas, en el primer caso el cifrado es simétrico o de llave privada y en el segundo es asimétrico o de llave pública. Actualmente estas técnicas criptográficas son la base de los mecanismos de seguridad [47]. La criptografía de llave privada se utiliza para cifrar grandes cantidades de datos y la criptografía de llave pública se prefiere para acordar una llave de sesión[1] que será utilizada con el método de cifrado simétrico.

1.2.1. Sistemas criptográficos simétricos o de llave privada

Los algoritmos de llave privada, están basados en el modelo mostrado en la Figura 1.2. Ellos utilizan una llave común con la cual es posible cifrar y descifrar los mensajes enviados entre dos o más partes. La fortaleza de estos algoritmos radica principalmente en que la llave sólo es conocida por las entidades que desean transmitir la información de manera confidencial. Sin embargo, uno de sus principales problemas con este tipo de esquemas es la distribución de la llave secreta. Los tamaños de las llaves utilizadas para cifrar/descifrar van desde los 64 bits (aunque actualmente se necesitan al menos 80 bits para considerar a una llave realmente segura) hasta los 256 bits. El grado de seguridad es determinado esencialmente por el tamaño de bits de la llave privada y el diseño del cifrador.

Las principales ventajas y desventajas del esquema de cifrado simétrico son:

- Ventajas:
  - Los cifradores simétricos pueden diseñarse para cifrar grandes cantidades de información
  - Las llaves usadas son más pequeñas que las usadas en cifradores asimétricos
  - Pueden emplearse como primitivas para la construcción de varios mecanismos criptográficos incluyendo generadores de números pseudoaleatorios, funciones picadillo (hash functions) y esquemas eficientes de firma digital

[1] Una llave de sesión es una llave de cifrado temporal para la comunicación entre dos sistemas.

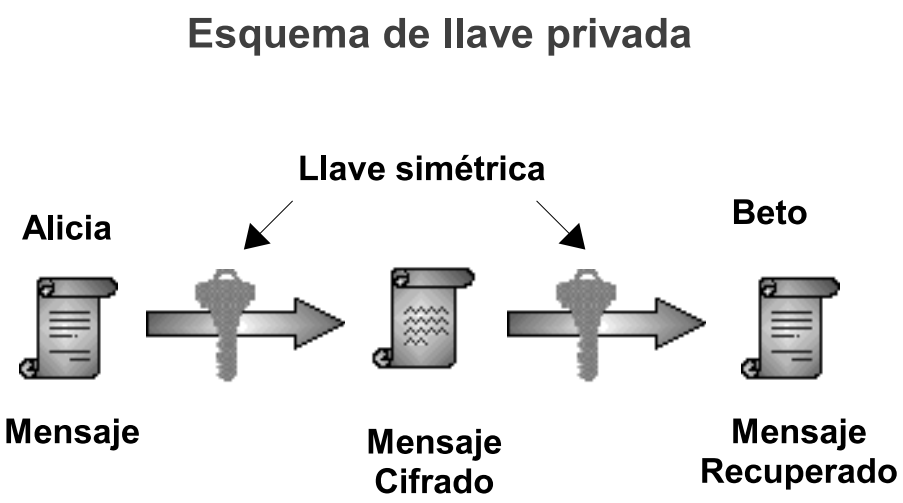

Figura 1.2: Esquema de cifrado simétrico o de llave privada

  - Pueden combinarse con otros bloques criptográficos en diversos modos de operación que permiten ofrecer conceptos de seguridad más amplios y robustos para producir cifradores más fuertes

- Desventajas:

  - La llave debe ser distribuida en secreto en un esquema de comunicación entre dos entidades
  - Es necesario tener un buen esquema de administración de llaves. Se necesitan $\frac{n(n-1)}{2}$ llaves secretas en un sistema con n usuarios
  - Es recomendable cambiar frecuentemente la llave usada en una comunicación entre dos entidades. Lo ideal es que exista una llave para cada sesión que se establezca

1.2.2. Sistemas criptográficos asimétricos o de llave pública

En este tipo de sistemas existen dos llaves distintas: una para cifrar y otra para descifrar. La llave para cifrar es conocida públicamente y, por ende, se denomina llave pública. La llave para descifrar sólo es conocida por el receptor del mensaje, por lo que se denomina llave privada. La llave pública puede ser usada por cualquier persona para cifrar mensajes bajo la premisa que sólo quien posea la llave privada podrá descifrar dichos mensajes. Ambas llaves están relacionadas matemáticamente pero debe ser computacionalmente imposible conocer la llave privada a partir de la llave pública.

Los algoritmos de llave pública evitan la necesidad de que exista previamente un secreto entre dos entidades que quieran comunicarse, pero son computacionalmente más demandantes que los algoritmos de llave secreta. Esto se debe a la gran cantidad de

cálculos que son necesarios para su implementación, lo cual los hace mucho más lentos. Por lo tanto, este tipo de algoritmos se utilizan en aplicaciones donde únicamente se requiere cifrar cantidades pequeñas de datos. La Figura 1.3 describe éste esquema.

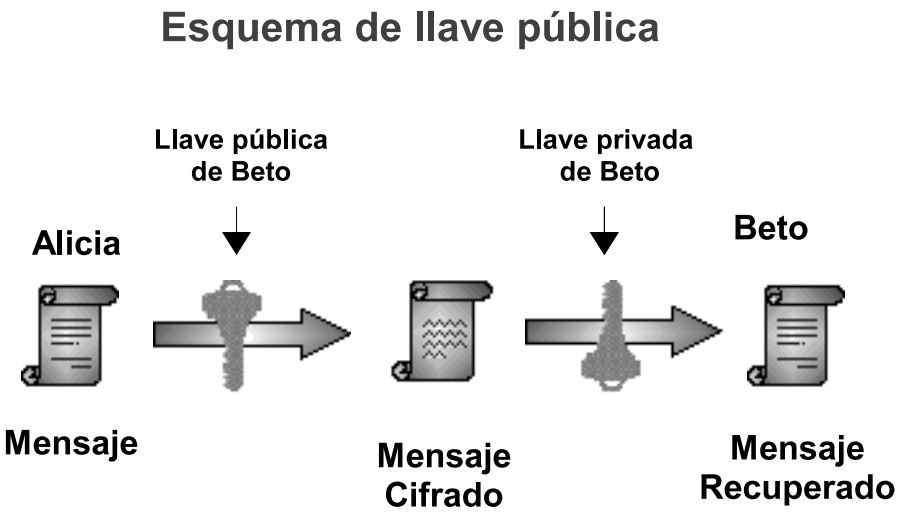

Figura 1.3: Esquema de cifrado asimétrico o de llave pública

Las principales ventajas y desventajas del esquema de cifrado asimétrico son:

- Ventajas:
  - Solamente la llave privada debe mantenerse en secreto, sin embargo, la autenticidad de las llaves públicas debe estar garantizada
  - La administración de las llaves en una red requiere la presencia de una tercera entidad de confianza y dependiendo de las exigencias de la red, ésta puede estar fuera de línea o en tiempo real
  - Dependiendo del modo de uso, las llaves privada y pública pueden ser utilizadas por periodos de tiempo extensos, por ejemplo, varias sesiones o incluso años
  - Muchos esquemas de llave pública ofrecen mecanismos eficientes de firma digital. La llave usada para la función pública de verificación es mucho más pequeña que la usada por los esquemas de llave simétrica
  - En una red grande, el número de llaves necesarias puede ser considerablemente más pequeño que en un escenario simétrico
- Desventajas:
  - El tiempo de ejecución de los métodos de cifrado de llave pública son considerablemente más lentos que los esquemas de llave secreta

- El tamaño de las llaves son mucho más grandes que las requeridas por los cifradores simétricos
- El tamaño de la firma digital usando el esquema de llave pública es grande en comparación con las etiquetas de autenticación de datos que se provee con las técnicas de llave simétrica

Ejemplos de estos algoritmos son RSA (Rivest-Shamir-Adleman) [40], ElGamal [15], DSA (Digital Signature Algorithm) y CCE (Criptografía de Curvas Elípticas) [29]. A continuación se mencionan dos tipos de esquemas criptográficos de llave pública con carácteristicas muy particulares.

Criptografía de curvas elípticas

Como ya se mencionó en §1.2.1 (en la página, 5) los sistemas criptográficos de llave privada son más eficientes que los de llave pública pero presentan, entre otros, el problema de distribución de llave. Debido parcialmente a ello existe una creciente tendencia a investigar nuevas técnicas que sean más eficientes para la implementación de criptografía de llave pública. En particular se ha incrementado el interés en criptografía de curvas elípticas CCE, ya que se ha visto que ésta tiene un costo computacional considerablemente menor que el esquema otros esquemas de cifrado asimétrico como RSA.
Los criptosistemas basados en curvas elípticas fueron propuestos por N. Koblitz [28] y V. Miller [34], y fue a partir de estas propuestas que se han realizado un sin número de estudios y análisis. Actualmente estos criptosistemas son ampliamente empleados en diversas aplicaciones.

Sistemas criptográficos basados en emparejamientos

El emparejamiento de Tate y el de Weil se introducen en la criptografía de manera independiente por Menezes [31] y Frey [18] como herramienta para criptoanálisis, empleados para reducir la complejidad del problema del logaritmo discreto en algunas curvas elípticas definidas sobre campos finitos.
Debido al descubrimiento de las propiedades constructivas de los emparejamientos por Joux [25], Sakai [45] y Mitsunari [37] es que éstos han sido aceptados como una herramienta para el diseño de protocolos, entre los cuales están: criptografía basada en identidad [8] y esquemas de firmas cortas [9].

Criptografía basada en identidad

En 1984 Adi Shamir [48] presentó este esquema criptográfico el cual permite que un par de usuarios se comuniquen de manera segura. La diferencia con las técnicas descritas anteriormente, es que los usuarios pueden verificar sus firmas sin la necesidad de intercambiar alguna llave (ya sea pública o secreta) ni guardar directorios de llaves y sin necesitar servicios de una tercera entidad.

Este esquema, como se muestra en la Figura 1.4, asume la existencia de un centro de generación de llaves confiable el cual proporcionará una tarjeta inteligente con la llave privada que se calcula a partir de cualquier identificador del usuario como su nombre, correo electrónico, teléfono, dirección, etc. dependiendo del contexto; con esta tarjeta el usuario es capaz de cifrar y firmar los mensajes que envíe así como descifrar y verificar los mensajes reciba.

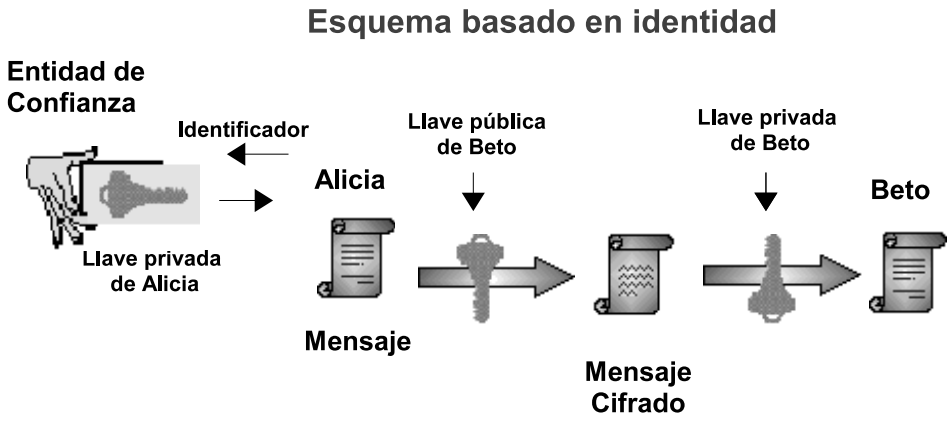

Figura 1.4: Esquema de cifrado basado en identidad

1.2.3. Nivel de seguridad

Los principales factores que determinan la seguridad de un algoritmo de cifrado simétrico por bloques son: la calidad del algoritmo mismo, el tamaño de llave utilizada y el tamaño de bloque que maneje dicho algoritmo.

Un algoritmo criptográfico común que tenga una llave de $m - bits$ pero cuya seguridad sea comparable con la de un algoritmo de cifrado simétrico por bloques que use una llave de $n - bits$, se dice que tiene una seguridad de $n - bits$. En general la seguridad equivalente de $n - bits$ de un algoritmo dado suele ser menor que m ya que existe la posibilidad de que exista algún ataque específico para dicho algoritmo que ofrezca ventajas computacionales con respecto al ataque de fuerza bruta.

Por otro lado, debido a razones de desempeño, funcionalidad o compatibilidad, frecuentemente se combinan algoritmos que cuentan con diferentes tamaños de llave en una misma aplicación. En general el algoritmo más débil y el tamaño de la llave utilizada son los que determinarán la seguridad que provee la aplicación. Por ejemplo, si se combina el algoritmo $SHA - 512$ con $RSA - 1024$ la aplicación proveerá un nivel de seguridad de $80 - bits$.

En la tabla 1.1 se presenta una comparación de los niveles de seguridad de un grupo de dos tipos de algoritmos criptográficos: de llave privada y llave pública [42].

Tabla 1.1: Comparación de niveles de seguridad [42]

Algoritmo de llave privada	**Seguridad** (bits)
Triple DES dos-llaves	80
Triple DES tres-llaves	112
AES $128 bits$	128
AES $192 bits$	192
AES $256 bits$	256
Algoritmo de llave pública	**Seguridad** (bits)
DSA ($p = 1024, q = 160$)	80
DSA ($p = 2048, q = 224$)	112
DSA ($p = 3072, q = 256$)	128
DSA ($p = 7680, q = 384$)	192
DSA ($p = 15360, q = 512$)	256
RSA $1024 bits$	80
RSA $2048 bits$	112
RSA $3072 bits$	128
RSA $7680 bits$	192
RSA $15360 bits$	256
ECC$\{160 - 223\} bits$	80
ECC$\{224 - 255\} bits$	112
ECC$\{256 - 383\} bits$	128
ECC$\{384 - 511\} bits$	192
ECC$\{512\} bits$	256

En las llaves simétricas la razón de utilizar 128 bits se debe principalmente a que se consideran llaves seguras de acuerdo con la ley de Moore que dice durará hasta 100 años sin romperse por fuerza bruta. Actualmente, un algoritmo de cifrado simétrico que cuente con una llave de 80 bits se considera, por lo general seguro, pero dado el aumento en el poder de cómputo que se vive día a día, éstos sistemas serán rotos aproximadamente en 2010; los que cuenten con llaves de 112 bits en 2030 y los de 128 bits en años posteriores a 2030 [42].

Capítulo 2
Preliminares matemáticos

Uno de los sustentos teóricos más importantes de la criptografía de curvas elípticas son los campos finitos o campos de Galois. Los campos finitos son estructuras algebraicas donde se definen las operaciones aritméticas y las propiedades que deben cumplir los puntos que forman las curvas elípticas. En este capítulo se presenta una breve introducción a las estructuras algebraicas necesarias para definir los campos. De igual forma, se analiza la aritmética definida en los campos, con énfasis en la multiplicación ya que es la operación más recurrente e importante. Además se realiza una breve descripción de los emparejamientos que son la principal aplicación en el desarrollo de esta tesis. Una descripción más detallada de estos temas puede ser consultada en [13] y [33].

2.1. Conceptos

2.1.1. Grupo

Un grupo $\langle G, * \rangle$ consiste de un conjunto G con una operación binaria $*$ sobre G que satisface los siguientes axiomas:

- La operación del grupo es asociativa $(a * b) * c = a * (b * c)$ $\forall a, b, c \in G$.
- Existe un elemento identidad denotado por $1 \in G$ tal que $a * 1 = 1 * a = a$.
- Para todo elemento $a \in G^*$ existe un elemento inverso multiplicativo $a^{-1} \in G^*$ tal que $a * a^{-1} = a^{-1} * a = 1$.

Un grupo G es conmutativo si $a * b = b * a$ $\forall a, b \in G$; también se conoce como grupo abeliano.

2.1.2. Anillo

Un anillo $\langle \mathcal{R}, +, \times \rangle$ está conformado por un conjunto $\mathcal{R}$ con dos operaciones definidas en sus elementos, aquí denotados por + y $\times \in \mathcal{R}$ que satisfacen las siguientes propiedades:

1. La estructura $\langle \mathcal{R}, + \rangle$ es un grupo abeliano con identidad denotada por 0.

2. La operación $\times$ es asociativa sobre $\mathcal{R}$: $a \times (b \times c) = (a \times b) \times c \ \forall a, b, c \in \mathcal{R}$

3. Existe la identidad multiplicativa denotada por 1 tal que $1 \times a = a \times 1 = a \ \forall a \in \mathcal{R}$.

4. La operación $\times$ es distributiva sobre +: $a \times (b + c) = (a \times b) + (a \times c)$ y $(b + c) \times a = (b \times a) + (c \times a) \ \forall a, b, c \in \mathcal{R}$.

Se dice que un anillo es conmutativo si $a \times b = b \times a \ \forall a, b \in \mathcal{R}$

2.1.3. Campos

Un campo es un anillo conmutativo, tal que todo elemento diferente de 0 tiene un inverso multiplicativo.

La característica de un campo es 0 si $\overbrace{1 + 1 + \ldots + 1}^{m \text{ veces}}$ nunca es igual a cero para cualquier $m \geq 1$. Por otro lado la característica de un campo es m si $\sum_{i=1}^{m} 1 = 0$.

Si la característica de un campo es diferente de cero, entonces m necesariamente es un número primo [33].

2.1.4. Campos finitos

Un campo finito o campo de Galois denotado por $\mathbb{F}_{q=p^n}$, es un campo de característica p y un número finito de elementos q. El orden del campo está dado por q. Un campo finito existe para cada número primo p y un entero n positivo, y contiene un subcampo con p elementos $\mathbb{F}_p$ conocido como campo base del campo original. Un campo finito también puede ser denotado por $\mathbb{F}_{p^n}$.

El orden de un campo finito está definido por el número de elementos que contenga. Existe un campo finito $\mathbb{F}$ de orden q siempre que q sea una potencia prima, esto es $q = p^m$ donde p es un número primo conocido como la característica del campo $\mathbb{F}$ y m es un entero positivo. Si $m = 1$ entonces $\mathbb{F}$ es llamado campo primo. Si $m \geq 2$ el nombre que recibe $\mathbb{F}$ es campo de extensión.

Para cualquier potencia prima q, existe en esencia un sólo campo finito de orden q, es decir que aunque las etiquetas para representar a los campos son diferentes cualesquiera campos finitos de orden q son estructuralmente el mismo. Por lo anterior se dice que estos campos son isomorfos y se denotan por $\mathbb{F}_q$.

Sea p un número primo, $m \geq 2$, $\mathbb{F}_p[x]$ el conjunto de todos los polinomios en la variable x con coeficientes en $\mathbb{F}_p$ y $f(x)$ el polinomio irreducible de grado m sobre $\mathbb{F}_p$ la extensión del campo puede ser definida como:

$$\mathbb{F}_{p^m} \cong \mathbb{F}_p[x]/\left(f(x)\right) \ \textit{tal que} \ b^i \neq 1$$

Los elementos en $\mathbb{F}_{p^m}$ son polinomios con coeficientes en $\mathbb{F}_p[x]$ de grado a lo más de $m-1$:

$$\mathbb{F}_{p^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0 : a_i \in \mathbb{F}_p\}$$

Los elementos diferentes de cero del campo finito $\mathbb{F}_p$, denotados por $\mathbb{F}_p^*$ forman un grupo cíclico multiplicativo. Por lo tanto existen elementos b denominados *generadores* tales que:

$$\mathbb{F}_p^* = \{b^i : 0 \leq i \leq p-2\}$$

El orden del elemento $a \in \mathbb{F}_p^*$ es el entero positivo más pequeño t que satisface $a^t = 1$. Ya que $\mathbb{F}_p^*$ es un grupo cíclico entonces t debe ser un divisor de $p-1$.

Un campo $\mathbb{F}$ tiene dos operaciones principales, suma y multiplicación. La resta de los elementos de un campo está definida en términos de la suma: $\forall\ a, b \in \mathbb{F}, a - b = a + (-b)$ donde $-b$ es conocido como el elemento inverso aditivo de b y es el único elemento en $\mathbb{F}$ tal que $b + (-b) = 0$. De manera análoga la división de elementos de un campo está definida en términos de la multiplicación: $\forall\ a, b \in \mathbb{F}$ con $b \neq 0, a/b = a \cdot b^{-1}$ donde b^{-1} es el único elemento en $\mathbb{F}$ tal que $b \cdot b^{-1} = 1$ y es denominado elemento inverso multiplicativo de b.

En criptografía los casos más utilizados son $q = p$, con p un número primo, $q = 2^m$ y $q = 3^m$. $\mathbb{F}_{2^m}$, es conocido como un campo finito de característica 2 o simplemente como campo de extensión binaria, sus elementos son cadenas de m bits. Las reglas para la aritmética en $\mathbb{F}_{2^m}$ pueden ser determinadas tanto por representación polinomial. Dado que las operaciones en este campo son en cadenas de bits, las computadoras las pueden realizar de manera muy eficiente y en hardware también presentan una ventaja ya que en $\mathbb{F}_{2^m}$ sólo se utilizan los valores de 0 y 1. Sin embargo recientemente la aritmética sobre $\mathbb{F}_{3^m}$ o campos de extensión ternaria han ganado gran importancia en varias aplicaciones criptográficas especialmente en criptografía de curvas elípticas y en criptografía basada en emparejamientos ya que se estima que la elección de una curva elíptica en el campo $\mathbb{F}_{3^m}$, ofrece una seguridad similar a una curva en $\mathbb{F}_{2^{m'}}$ [38], donde:

$$m' = m(\log_2 3) \approx m(1.58)$$

2.1.4.1. Campos finitos binarios

Reciben el nombre de campos binarios los campos finitos con orden 2^m. Los elementos que pertenecen a $\mathbb{F}_{2^m}$ son de grado a lo más $m-1$ y pueden ser representados de forma polinomial con coeficientes definidos en el campo $\mathbb{F}_2 = \{0, 1\}$:

$$\mathbb{F}_{2^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0 : a_i \in \{0, 1\}\}$$

Sea $f(x)$ un polinomio binario irreducible de grado m, es decir, que no puede ser factorizado como un producto de polinomios binarios de grado menor a m. La suma de los elementos de un campo se lleva a cabo como una suma de polinomios considerando

que las operaciones entre los coeficientes deben ser módulo 2 y la multiplicación se debe realizar módulo el polinomio irreducible $f(x)$. Para cualquier polinomio binario $a(x)$ la operacion $a(x)$ mod $f(x)$ deberá resultar en un polinomio residuo $r(x)$ de grado menor a m que se obtiene a través de la división de $a(x)$ entre $f(x)$; esta operación es conocida como reducción modular. Las operaciones aritméticas en este campo se describen en §2.2 (en la página 14).

2.1.4.2. Campos finitos ternarios

La representación en base polinomial para los campos binarios puede ser generalizada para cualquier extensión de un campo, cada coeficiente (trit) $a \in \mathbb{F}_3$ puede ser codificado como 2 bits a^1, a^0 con $a = 2a^1 + a^0$. En el campo finito $\mathbb{F}_{3^m}$ los coeficientes a_i pertenecen a $\mathbb{F}_3$ y pueden ser representados utilizando un par de bits (a_i^1, a_i^0) como se recomienda en [23] donde:

$$\{0 \to (0,0), 1 \to (0,1), 2 \to (1,0)\}$$

Cuando ambos bits se encuentran encendidos se considera un estado inválido. La negación de un elemento en $\mathbb{F}_3$ utilizando esta representación es sencilla ya que únicamente es necesario invertir los bits:

$$-(a_i^1, a_i^0) = (a_i^0, a_i^1)$$

Teniendo la ventaja de que la resta se puede implementar a partir de la suma, esto es porque los inversos aditivos quedan definidos cambiando la posición de los bits, teniendo que:

$$-1 = 2$$
$$-2 = 1$$

Es decir, restar uno equivale a sumar dos y restar dos equivale a sumar uno.

En §2.2 se dará la descripción de las operaciones aritméticas en este campo.

2.2. Aritmética en campos $\mathbb{F}_{2^m}$ y $\mathbb{F}_{3^m}$

En esta sección se definen las operaciones de suma, resta, multiplicación así como el operador de Frobenius y el operador inverso de Frobenius en $\mathbb{F}_{2^m}$ y $\mathbb{F}_{3^m}$ de las cuales se detalla la multiplicación polinomial ya que es el tema principal para el desarrollo de esta tesis.

2.2.1. Suma y resta

Las operaciones de suma y resta en $\mathbb{F}_2$ son equivalentes dado que el inverso aditivo de un elemento es el elemento mismo; en $\mathbb{F}_{2^m}$ estas operaciones pueden ser implementadas utilizando una operación XOR entre dos operandos de m bits.

Tabla 2.1: Definición de la suma en $\mathbb{F}_3$.

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Para un elemento $a \in \mathbb{F}_3$ las operaciones de suma y resta quedan definidas en las tablas 2.1 y 2.2 respectivamente.

Tabla 2.2: Definición de la resta en $\mathbb{F}_3$.

−	0	1	2
0	0	2	1
1	1	0	2
2	2	1	0

Sean $A, B\ y\ C \in \mathbb{F}_3$ donde:

$$\begin{aligned} A &= A_1 \| A_0 \\ B &= B_1 \| B_0, \end{aligned} \tag{2.1}$$

la operación de suma en $\mathbb{F}_3$ a nivel de compuertas se define de la siguiente manera [23]:

$$\begin{aligned} C &= A + B \\ C_0 &= (A_1 \text{ xor } B_1) \text{ or } t \\ C_1 &= (A_0 \text{ xor } B_0) \text{ or } t \\ t &= (A_0 \text{ xor } B_1) \text{ xor } (A_1 \text{ xor } B_0) \end{aligned}$$

2.2.2. Multiplicación

Para un elemento $a \in \mathbb{F}_3$ las operacion de multiplicación esta definida como se presenta en la tabla 2.3.

Sean $A, B\ y\ C \in \mathbb{F}_3$ donde:

$$\begin{aligned} A &= A_1 \| A_0 \\ B &= B_1 \| B_0, \end{aligned} \tag{2.2}$$

Tabla 2.3: Definición del producto en $\mathbb{F}_3$.

$*$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

se define la operación de multiplicacón en $\mathbb{F}_3$ a nivel de compuertas como sigue [23]:

$$\begin{aligned} C &= A * B \\ C_0 &= (A_0 \, and \, B_0) \, or \, (A_1 \, and \, B_1) \\ C_1 &= (A_0 \, and \, B_1) \, or \, (A_1 \, and \, B_0) \end{aligned}$$

En §2.2.8 se presenta un análisis detallado de la multiplicación polinomial para campos finitos ya que es la operación fundamental para el desarrollo de esta tesis.

2.2.3. Elevar al cuadrado

Por otro lado las operaciones de raíz cuadrada y elevar al cuadrado se han vuelto muy importantes ya que son indispensables para el diseño de algunas primitivas de curvas elípticas. En [41] se presentan algoritmos para el cálculo de dichas operaciones en campos finitos binarios que se construyan utilizando trinomios irreducibles de la forma $P(x) = x^m + x^n + 1$ con $m \geq 2$, a continuación se explica la operación de elevar al cuadrado.

Sea $A = \sum_{i=0}^{m-1} a_i x^i$ un elemento arbitrario de $\mathbb{F}_{2^m}$. Y

$$A^2(x) = \left(\sum_{i=0}^{m-1} a_i x^i \right) \left(\sum_{i=0}^{m-1} a_i x^i \right) = \sum_{i=0}^{m-1} a_i x^{2i} \tag{2.3}$$

de acuerdo a esta ecuación 2.3 dicho cuadrado, A^2, puede ser representado por un vector de $2m$ coeficientes.

$$\begin{aligned} A^2(x) &= [0 \; a_{m-1} \; 0 \; a_{m-2} \cdots 0 \; a_1 \; 0 \; a_0] \\ &= [a'_{2m-1} \; a'_{m-2} \cdots \; a'_{m-1} \; a'_m ; a'_{m-1} \; a'_2 \cdots a'_1 \; a'_0] \end{aligned}$$

donde $a'_i = 0$ para todos los valores de i impares. Los autores de [41] reducen dicho vector utilizando la regla de reducción:

$$k \geq m \Rightarrow x^k = x^{k-m} x^m = x^{k-m}(1 + x^n) = x^{k-m} + x^{k-m+n}$$

Y obtienen que la primera mitad más significativa de A^2, es decir, los m bits más significativos del vector anterior se pueden proyectar a las primeras m coordenadas

realizando únicamente sumas y corrimientos. Los autores categorizan a todos los trinomios irreducibles en cuatro tipos diferentes:

Tipo I	m par, n impar	$n < \frac{m}{2}$
Tipo II	m par, n impar	$n = \frac{m}{2}$
Tipo III	m, n impares	$n < \frac{m+1}{2}$
Tipo IV	m impar, n par	$n < \frac{m+1}{2}$

En esta tesis se explicará únicamente el tipo III de acuerdo a los trinomios irreducibles utilizados, la explicación detallada de todos los tipos se encuentra en [41].

Tipo III: Cálculo de $C = A^2 \bmod P(x)$ con $P(x) = x^m + x^n + 1$, m, n números impares y $n < \frac{m+1}{2}$.

$$c_i = \begin{cases} a_{\frac{i}{2}} & i \; par, i < n \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+\frac{m-n}{2}} + a_{\frac{i}{2}+(m-n)} & i \text{ par}, n < i < 2n, \\ a_{\frac{i}{2}} + a_{\frac{i}{2}+\frac{m-n}{2}} & i \text{ par}, i \geq 2n, \\ a_{\frac{m+i}{2}} + a_{\frac{m+i}{2}+\frac{m-n}{2}} & i \text{ impar}, i < n, \\ a_{\frac{m+i}{2}} & i \text{ impar}, i \geq n \end{cases}$$

para $i = 0, 1, \cdots, m-1$.

2.2.4. Raíz cuadrada

Considerando que el coeficiente n del polinomio $P(x) = x^m + x^n + 1$ satisface que $1 \leq n \leq \frac{m}{2}$, calcular la raíz cuadrada de un elemento arbitrario A del campo significa encontrar un elemento del campo $D = \sqrt{A}$ tal que $D^2 = MD = A$ por lo tanto,

$$D = M^{-1}A$$

Esta ecuación es especialmente atractiva para los campos finitos binarios con orden suficientemente grande. De la misma manera que en el cálculo de la operación de elevar al cuadrado los autores clasifican a los trinomios irreducibles en cuatro tipos:

Tipo I	m par, n impar	$n < \frac{m}{2}$
Tipo II	m par, n impar	$n = \frac{m}{2}$
Tipo III	m, n impares	$n < \frac{m+1}{2}$
Tipo IV	m impar, n par	$\lceil \frac{m-1}{4} \rceil \leq n < \lfloor \frac{m-1}{3} \rfloor$

Análogamente con la operación de elevar al cuadrado, se detallará el tipo III de acuerdo a los trinomios irreducibles utilizados, ver [41].

Tipo III: Cálculo de D tal que $D^2 = A \bmod P(x)$ con $P(x) = x^m + x^n + 1$, m, n números impares y $n < \frac{m+1}{2}$.

$$d_i = \begin{cases} a_{2i} & i < \frac{n+1}{2} \\ a_{2i} + a_{2i-n} & \frac{n+1}{2} \leq i < \frac{m+1}{2} \\ a_{2i-n} + a_{2i-m} & \frac{m+1}{2} \leq i < \frac{m+n}{2} \\ a_{2i-m} & \frac{m+n}{2} \leq i < m \end{cases}$$

para $i = 0, 1, \cdots, m-1$.

2.2.5. Elevar al cubo

Respecto a característica tres, un elemento $A \in \mathbb{F}_{3m}$ se puede representar en base canónica así [1]:

$$A = \sum_{i=0}^{m-1} a_i x^i = \sum_{i=0}^{u-s} a_{3i} x^{3i} + x \sum_{i=0}^{u+r-2} a_{3i+1} x^{3i} + x^2 \sum_{i=0}^{u-1} a_{3i+2} x^{3i} \tag{2.4}$$

Sea A un elemento arbitrario en $\mathbb{F}_{3^m}$, $m = 3u + r$ donde $r \in \mathbb{F}_3$; $r \in [-1, 1]$ el cubo de A denotado por A^3 es el elemento $C \in \mathbb{F}_{3^m}$ tal que $C = A^3$ y puede ser calculado de la siguiente manera:

$$\begin{aligned} A^3 &= \left(\sum_{i=0}^{m-1} a_i x^i \right)^3 = \sum_{i=0}^{m-1} a_i x^{3i} \\ &= \sum_{i=0}^{u} a_i x^{3i} + \sum_{i=u+1}^{2u+r-1} a_i x^{3i} + \sum_{i=2u+r}^{3u+r-1} a_i x^{3i} \\ &= C_0 + x^{3u+r} C_1 + x^{6u+2r} C_2 \end{aligned}$$

$$\begin{aligned} A^3 &= \left(\sum_{i=0}^{m-1} a_i x^i \right)^3 = \sum_{i=0}^{m-1} a_i x^{3i} \\ &= \sum_{i=0}^{u} a_i x^{3i} + \sum_{i=u+1}^{2u+r-1} a_i x^{3i} + \sum_{i=2u+r}^{3u+r-1} a_i x^{3i} \\ &= C_0 + x^{3u+r} C_1 + x^{6u+2r} C_2 \end{aligned}$$

donde,

$$\begin{aligned} C_0 &= \sum_{i=0}^{u} a_i x^{3i} \\ C_1 &= \sum_{i=1}^{u+r-1} a_{i+u} x^{3i-r} \\ C_2 &= \sum_{i=r}^{u+r-1} a_{i+2u} x^{3i-2r} \end{aligned}$$

2.2.6. Raíz cúbica:

La raíz cúbica de A denotada por $\sqrt[3]{A}$ es el elemento $D \in \mathbb{F}_{3^m}$ tal que $D^3 = A$. De acuerdo a la notación de la ecuación 2.4 se tiene que $s = 1$ si $r = 1$ y $s = 0$ en cualquier otro caso. Entonces la raíz cúbica $\sqrt[3]{A}$ se puede calcular de la siguiente manera:

$$\sqrt[3]{A} = \sum_{i=0}^{u-s} a_{3i} x^i + x^{\frac{1}{3}} \sum_{i=0}^{u+r-2} a_{3i+1} x^i + x^{\frac{2}{3}} \sum_{i=0}^{u-1} a_{3i+2} x^i \bmod P(x)$$

2.2.7. Inversión

Existen diversos algoritmos para obtener el inverso multiplicativo de un elemento en el campo $\mathbb{F}_{2^m}$, en esta tesis nos enfocaremos al algoritmo de Itoh-Tsuiji [24] el cual se explica a continuación.

Puesto que el grupo multiplicativo del campo finito $\mathbb{F}_{2^m}$ es cíclico y de orden $2^m - 1$, entonces para cualquier elemento diferente de cero en $\mathbb{F}_{2^m}$ se tiene que: $a^{-1} = a^{2^m-2}$ ya que:

$$2^m - 2 = 2(2^{m-1} - 1) = 2 \sum_{j=0}^{m-2} 2^j = \sum_{j=1}^{m-1} 2j$$

Sea $\left(\beta_k(a) = a^{2^k-1}\right)_{k\in\mathbb{N}}$ entonces:

$$\beta_0(a) = 1, \;\; \beta_1(a) = a$$

y

$$\beta_{m-1}(a)^2 = a^{-1}$$

Para ilustrar el funcionamiento de este algoritmo, las tablas 2.4 y 2.5 muestran las operaciones necesarias para calcular el inverso multiplicativo de un elemento en $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{2^{313}}$. Empleamos cadenas de adición para calcular exponenciaciones para requerir el menor número de multiplicaciones necesarias. Una cadena de adición no es mas que una secuencia de números naturales donde cada término que se genera es la suma de dos terminos previos. En este trabajo consideramos aquellas cadenas de adición con el menor

número de iteraciones necesarias.

De acuerdo a [12] para $\mathbb{F}_{2^{239}}$ se utilizó la siguiente cadena de adición:

$$1 - 2 - 3 - 6 - 7 - 14 - 28 - 56 - 112 - 224 - 238$$

Sea $A \in \mathbb{F}_{2^{239}}$ la tabla 2.4 muestra las operaciones necesarias para encontrar A^{-1} mediante el método de Itoh Tsujii.

Tabla 2.4: Método Itoh-Tsujii para encontrar un inverso en $\mathbb{F}_{2^{239}}$

$\beta_0 =$	A
$\beta_1 =$	$\beta_0\beta_0^{2^1}$
$\beta_2 =$	$\beta_0\beta_1^{2^1}$
$\beta_3 =$	$\beta_2\beta_2^{2^3}$
$\beta_4 =$	$\beta_0\beta_3^{2^1}$
$\beta_5 =$	$\beta_4\beta_4^{2^7}$
$\beta_6 =$	$\beta_5\beta_5^{2^{14}}$
$\beta_7 =$	$\beta_6\beta_6^{2^8}$
$\beta_8 =$	$\beta_7\beta_7^{2^{56}}$
$\beta_9 =$	$\beta_8\beta_8^{2^{112}}$
$A^{-1} =$	$\beta_5\beta_9^{2^{14}}$

De acuerdo a [12] para $\mathbb{F}_{2^{313}}$ se utilizó la siguiente cadena de adición:

$$1 - 2 - 4 - 6 - 12 - 24 - 48 - 96 - 192 - 288 - 312$$

Sea $A \in \mathbb{F}_{2^{313}}$ la tabla 2.5 muestra las operaciones necesarias para encontrar A^{-1} mediante el método de Itoh Tsujii.

Encontrar un inverso multiplicativo usando el método de Itoh-Tsujii requiere del cálculo de arriba de $\frac{m-1}{2}$ elevaciones al cuadrado seguidas.

2.2.8. Multiplicación polinomial

La multiplicación es por mucho la operación aritmética que recibe mayor atención ya que los cálculos requeridos para computarla consumen una gran cantidad de tiempo, en especial cuando son realizados en software. Así que por razones de eficiencia y seguridad, es preferible implementar tales algoritmos en hardware. Gracias a la popularización de los sistemas de diseño VLSI (Very Large Scale Integration) y la aparición de circuitos de lógica programable actualmente es posible una migración rápida y económica de algoritmos hacia hardware. De esta forma, la implementación en FPGA (Field Programmable

Tabla 2.5: Método Itoh-Tsujii para encontrar un inverso en $\mathbb{F}_{2^{313}}$

$\beta_0 =$	A
$\beta_1 =$	$\beta_0\beta_0^{2^1}$
$\beta_2 =$	$\beta_1\beta_1^{2^2}$
$\beta_3 =$	$\beta_1\beta_2^{2^2}$
$\beta_4 =$	$\beta_3\beta_3^{2^6}$
$\beta_5 =$	$\beta_4\beta_4^{2^{12}}$
$\beta_6 =$	$\beta_5\beta_5^{2^{24}}$
$\beta_7 =$	$\beta_6\beta_6^{2^{48}}$
$\beta_8 =$	$\beta_7\beta_7^{2^{96}}$
$\beta_9 =$	$\beta_7\beta_8^{2^{96}}$
$A^{-1} =$	$\beta_5\beta_9^{2^{24}}$

Gate Array) se presenta como una opción atractiva para diseñadores, debido a su excelente relación costo/beneficio, alta flexibilidad, posibilidad para optimizar/reconfigurar diseños, etc.

Sean $A(x), B(x)$ y $C'(x) \in \mathbb{F}_{p^m}$ y $P(x)$ un polinomio irreducible de grado m. La multiplicación en $\mathbb{F}_{p^m}$ es definida como una multiplicación polinomial módulo el polinomio irreducible $P(x)$.

$$C'(x) = [A(x) * B(x)] \bmod P(x)$$

En la literatura abierta se han reportado distintas maneras de clasificar a los multiplicadores polinomiales, en [42] se clasifican en dos categorías dependiendo de su complejidad: cuadráticos y subcuadráticos.

Algunos ejemplos de multiplicadores cuadraticos son: el método clásico de la escuela, multiplicadores entrelazados, multiplicadores matriz-vector, multiplicador de Montgomery, multiplicadores basados en la regla de Horner. Por otro lado algunos ejemplos de multiplicadores subcuadráticos son: multiplicadores basados en el teorema del residuo chino, multiplicadores matriz-vector Toeplitz, multiplicador Winograd, multiplicador Karatsuba-Ofman y multiplicador Toom-Cook.

Esta última categoría ofrece bajo espacio de complejidad, especialmente para valores grandes de m. De forma general los multiplicadores presentados en esta tesis se pueden clasificar como multiplicadores de dos pasos, donde el primer paso consiste en obtener el

producto $C(x)$ de grado a lo más de $2m-2$ dado por:

$$C(x) = A(x) * B(x) = \left(\sum_{i=0}^{m-1} a_i x^i\right)\left(\sum_{i=0}^{m-1} b_i x^i\right) \tag{2.5}$$

El segundo paso es realizar una operación de reducción modular para así obtener el polinomio $C'(x)$ de grado $m-1$:

$$C'(x) = C(x) \bmod P(x)$$

Por otro lado existen diversos esquemas que permiten calcular:

$$C(x) = [A(x)B(x)] \bmod P(x)$$

En el esquema de multiplicación *serial*, un sólo dígito o coeficiente de $A(x)$ se procesa en cada iteración, de esta manera el resultado se obtiene después de ejecutar m iteraciones. Los multiplicadores *paralelos* calculan $A(x)B(x)$ como en (2.5) y se necesitará de la reducción modular, éstos ofrecen un alto rendimiento al costo de incrementar los recursos computacionales necesarios para su realización. Con el fin de aprovechar las ventajas de los dos esquemas mencionados anteriormente Song y Parhi en [30] presentaron el esquema *serial-paralelo* cuya idea principal consiste en procesar D dígitos o coeficientes del operando en cada iteración. D en ocasiones se referencía como el tamaño del dígito; cuando $D = 1$ se trata del esquema serial.

Existen muchos algoritmos para calcular la multiplicación polinomial, entre los más relevantes se encuentra el de Karatsuba-Ofman [26] que fue descubierto en 1962, es el primer método de multiplicación que rompe la barrera de complejidad cuadrática; por lo que es ampliamente utilizado para diseños en hardware reconfigurable y existe una gran cantidad de variantes que buscan mejorar los aspectos de área y tiempo de respuesta. A continuación se describen brevemente algunos de estos algoritmos.

2.2.8.1. Algoritmo Karatsuba-Ofman

Este algoritmo se utiliza para realizar la multiplicación de dos operandos en un campo finito determinado, brindando una complejidad de $\bigcirc(n^{\log_2 3})$ [26]; dicho algoritmo puede ser utilizado para los campos finitos binarios $\mathbb{F}(2^m)$ o en los campos finitos ternarios $\mathbb{F}(3^m)$. Los operandos pueden ser representados en base polinomial como:

$$\begin{aligned} A(x) &= \sum_{i=0}^{m-1} a_i x^i = \sum_{i=m/2}^{m-1} a_i x^i + \sum_{i=0}^{\lfloor m/2 \rfloor - 1} a_i x^i \\ &= x^{\lfloor m/2 \rfloor} \sum_{i=0}^{(m/2)-1} a_{i+(m/2)} x^i + \sum_{i=0}^{\lfloor m/2 \rfloor - 1} a_i x^i \end{aligned}$$

$$\begin{aligned} B(x) &= \sum_{i=0}^{m-1} b_i x^i = \sum_{i=m/2}^{m-1} b_i x^i + \sum_{i=0}^{\lfloor m/2 \rfloor - 1} b_i x^i \\ &= x^{\lfloor m/2 \rfloor} \sum_{i=0}^{(m/2)-1} b_{i+(m/2)} x^i + \sum_{i=0}^{\lfloor m/2 \rfloor - 1} b_i x^i \end{aligned}$$

Quedando A y B representados por $A = A_H || A_L$ y $B = B_H || B_L$ donde el sufijo H denota la palabra alta y el sufijo L la palabra baja.

$$\begin{aligned} A(x) &= x^{\lfloor m/2 \rfloor} A_H + A_L \\ B(x) &= x^{\lfloor m/2 \rfloor} B_H + B_L \end{aligned}$$

Usando esta notación, el producto polinomial está dado por:

$$C(x) = A_H B_H x^m + (A_H B_L + A_L B_H)\, x^{\frac{m}{2}} + A_L B_L \tag{2.6}$$

El algoritmo de Karatsuba-Ofman está basado en la idea de que el producto de la ecuación (2.6) se puede reescribir de la siguiente manera:

$$\begin{aligned} C(x) &= x^m A_H B_H + (A_H B_H + A_L B_L + (A_H + A_L)(B_L + B_H)) x^{\frac{m}{2}} + A_L B_L \\ &= x^m C_H + C_L \end{aligned} \tag{2.7}$$

$$\begin{aligned} M_A &= A_H + A_L \\ M_B &= B_H + B_L \\ M &= M_A M_B \end{aligned}$$

De acuerdo a la ecuacion (2.7) el producto polinomial $C(x)$ de grado a lo más $2m - 2$ puede verse como:

$$\begin{aligned} C_H &= [c_{2m-2}, c_{2m-3}, \ldots, c_{m+1}, c_m]; \\ C_L &= [c_{m-1}, c_{m-2}, \ldots, c_2, c_1, c_0] \end{aligned}$$

Aunque la ecuación (2.7) pareciera más complicada que la ecuación (2.6), se puede notar que (2.7) puede utilizarse para calcular el producto realizando 4 sumas y 3 multiplicaciones polinomiales a diferencia de (2.6) donde se necesitan 3 sumas y 4 multiplicaciones polinomiales. Se concluye que (2.7) es computacionalmente más eficiente ya que evaluar una multiplicación polinomial resulta mucho más costoso que realizar una suma.

Con este algoritmo se reduce el costo computacional al disminuir el número de multiplicaciones parciales que hay que realizar. El algoritmo Karatsuba-Ofman es un algoritmo recursivo y sigue la estrategia de *divide y vencerás*. Es muy eficiente para campos finitos $\mathbb{F}(p^m)$, donde m es una potencia de 2, sin embargo para fines criptográficos una condición deseable es escoger m como un número primo, en estos casos las implementaciones clásicas del algoritmo Karatsuba-Ofman contienen operaciones redundantes; es por ello que a partir de este diseño se han propuesto algunas variantes para mejorar la eficiencia del algoritmo como en [10] y [17] entre otros.

Algoritmo 1: Algoritmo Karatsuba-Ofman

Entrada: $A, B \in \mathbb{F}_{p^m}$, $r = m$
Salida: $C = AB$

if *r==1* **then**
    $C \leftarrow mul_n(A, B)$;
    return;
end
for $i \leftarrow 0$ **to** $r/2 - 1$ **do**
    $M_{A_i} \leftarrow A_i^L + A_i^H$;
    $M_{B_i} \leftarrow B_i^L + B_i^H$;
end
$mul(C^L, A^L, B^L)$;
$mul(M, M_A, M_B)$;
$mul(C^H, A^H, B^H)$;
for $i \leftarrow 0$ **to** $r - 1$ **do**
    $M_i \leftarrow M_i + C_i^L + C_i^H$;
end
for $i \leftarrow 0$ **to** $r - 1$ **do**
    $C_{r/2+i} \leftarrow C_{r/2+i} + M_i$;
end

2.2.8.2. Algoritmo Karatsuba-Ofman no redundante

N. Chang, C.Kim, Y. Park y J. Lim [10] proponen esta variante del algoritmo Karatsuba-Ofman, por sus siglas en ingles denominado NRKOA donde se busca reducir el área de las implementaciones en hardware evitando repetición de operaciones ya que observaron que dados $A(x), B(x) \in \mathbb{F}_{p^m}$ con $m = r + s, 0 < s < r = 2^k$, donde k es un entero positivo, el producto polinomial $C = AB$ puede calcularse utilizando una división no balanceada de los operandos de la siguiente manera:

$$A(x) = \sum_{i=0}^{m} a_i x^i \rightarrow A^L = \sum_{i=0}^{r-1} a_i x^i; A^H = x^r \sum_{i=0}^{s-1} a_{i+r} x^i$$

$$B(x) = \sum_{i=0}^{m} b_i x^i \rightarrow B^L = \sum_{i=0}^{r-1} b_i x^i; B^H = x^r \sum_{i=0}^{s-1} b_{i+r} x^i$$

entonces,

$$A^H + A^L = \sum_{i=0}^{s-1} (a_i + a_{i+r}) x^i + \sum_{i=s}^{m-1} a_i x^i;$$

$$B^H + B^L = \sum_{i=0}^{s-1} (b_i + b_{i+r}) x^i + \sum_{i=s}^{m-1} b_i x^i.$$

De esta manera los $m - s$ bits más significativos de los términos $(A^H + A^L)$ y A^L son idénticos, lo mismo ocurre para los términos $(B^H + B^L)$ y B^L. Entonces si los productos parciales $(A^L B^L)$ y $(A^H + A^L)(B^H + B^L)$ se calculan de manera cuidadosa, el producto de los $m - s$ bits más significativos puede ser compartido evitando así calcular el mismo producto dos veces. Debido a razones prácticas los autores sugieren usar este método cuando se cumpla la condición de que $s < 2^{k-1}$.

Por lo anterior, el algoritmo permite evitar las multiplicaciones redundantes en los cálculos, ocasionando una disminución en área con respecto a las implementaciones clásicas. Se basa principalmente en tres algoritmos que permiten el cálculo de la multiplicación de dos operandos, uno de éstos es el algoritmo clásico de multiplicación para cuatro bits; que es donde se termina la recursividad, los otros dos se presentan a continuación.

Algoritmo 2: Algoritmo Karatsuba-Ofman no redundante

Entrada: $a(x), b(x) \in (\mathbb{F}_{p^m})$
Salida: $c(x) = a(x) \cdot b(x)$

if $n < 7$ **then**
    Mul $(c(x), a(x), b(x), n)$;
    return $(c(x))$;
end
$c(x) \leftarrow 0$;
$m_1 \leftarrow 2^{\lfloor \log(n-1) \rfloor}$ y $m_2 \leftarrow n - m_1$;
for i **from** 0 **to** $m_2 - 1$ **do**
    $ADD_{a(x)} \leftarrow a_i + a_{m+i}$;
    $ADD_{b(x)} \leftarrow b_i + b_{m+i}$;
end
for i **from** m_2 **to** $m_1 - 1$ **do**
    $ADD_{a(x)} \leftarrow a_i$;
    $ADD_{b(x)} \leftarrow b_i$;
end
$NRHKOA\left(c(x)_0^{2m-2}, a(x)_0^{m-1}, b(x)_0^{m-1}, t(x)_0^{2m-2}, ADD_{a(x)}, ADD_{b(x)}, m_1, m_2\right)$; // Alg.3
$NRKOA\left(c(x)_{2m}^{2n-2}, a(x)_m^{n-1}, b(x)_m^{n-1}, m_2\right)$; // Alg.2
$c(x)_m^{m+n-2} \leftarrow c(x)_m^{m+n-2} + \left(t(x)_0^{n-2} + c(x)_0^{n-2} + c(x)_2^{2n-2}\right) x^m$;
return $c(x)$;

En el algoritmo 3 se realizan al mismo tiempo dos productos, en los cuales se evita realizar operaciones redundantes. Para observar la redundancia de algunos términos en las multiplicaciones basta con un ejemplo como lo es el producto de polinomios de grado dos.

Ejemplo Sean $A(x), B(x) \in \mathbb{F}_{3^3}$ donde:

Algoritmo 3: Algoritmo medio Karatsuba-Ofman no redundante

Entrada: $a_1(x), b_1(x), a_2(x), b_2(x) \in (\mathbb{F}_{p^m}), m_1, m_2$
Salida: $c(x) = a_1(x) \cdot b_1(x), d(x) = a_2(x) \cdot b_2(x)$

if $m = 4$ **then**
    $Mul4bit\ (c(x), a_1(x), b_1(x), m_2, 1)$;
    $Mul4bit\ (d(x), a_2(x), b_2(x), m_2, 0)$;
    return $(c(x), d(x))$;
end
Set $f \leftarrow \min\left(\frac{m_1}{2}, m_2\right)$;
$g \leftarrow \max\left(m_2 - \frac{m_1}{2}, 0\right)$;
for i **from** 0 **to** $\frac{m_1}{2} - 1$ **do**
    $ADD_{a_1(x)} \leftarrow a_{1i} + a_{1(m_1/2)+i}$;
    $ADD_{b_1(x)} \leftarrow b_{1i} + b_{1(m_1/2)+i}$;
end
for i **from** 0 **to** $f - 1$ **do**
    $ADD_{a_2(x)} \leftarrow a_{2i} + a_{2(m_1/2)+i}$;
    $ADD_{b_2(x)} \leftarrow b_{2i} + b_{2(m_1/2)+i}$;
end
for i **from** f **to** $\frac{m_1}{2} - 1$ **do**
    $ADD_{a_2(x)} \leftarrow ADD_{a_1(x)}$;
    $ADD_{b_2(x)} \leftarrow ADD_{b_1(x)}$;
end
$NRHKOA\left(c(x)_0^{m_1-1}, a_1(x)_0^{m_1/2-1}, b_1(x)_0^{m_1/2-1}, d(x)_0^{m_1-1}, a_2(x)_0^{m_1/2-1}, b_2(x)_0^{m_1/2-1}, \frac{m1}{2}, f\right)$;
// Alg. 3
$NRHKOA\left(u(x), ADD_{a_1(x)}, ADD_{b1(x)}, v(x), ADD_{a_2(x)}ADD_{b_2(x)}, \frac{m1}{2}, f\right)$; // Alg. 3
if $g = 0$ **then**
    $KOA\left(c(x)_{m_1}^{2m_1-2}, a_1(x)_{m_1/2}^{m_1-1}, b_1(x)_{m_1/2}^{m_1-1}, \frac{m1}{2}\right)$; // Alg. 1
    $d(x)_{m_1}^{2m_1-2} \leftarrow c(x)_{m_1}^{2m_1-2}$;
else
    $NRHKOA\left(c(x)_{m_1}^{2m_1-2}, a_1(x)_{m_1/2}^{m_1-1}, b_1(x)_{m_1/2}^{m_1-1}, d(x)_{m_1}^{2m_1-2}, a_2(x)_{m_1/2}^{m_1-1}, b_2(x)_{m_1/2}^{m_1-1}, \frac{m1}{2}, g\right)$;
    // Alg. 3
end
$c(x)_{m_1/2}^{3m_1/2-2} \leftarrow c(x)_{m_1/2}^{3m_1/2-2} + u(x) + c(x)_0^{m_1-2} + c(x)_{m_1}^{2m_1-2}$;
$d(x)_{m_1/2}^{3m_1/2-2} \leftarrow d(x)_{m_1/2}^{3m_1/2-2} + v(x) + d(x)_0^{m_1-2} + d(x)_{m_1}^{2m_1-2}$;
return $(c(x), d(x))$;

$$A(x) = a_0 + a_1 x + a_2 x^2 \therefore A_L = a_0 + a_1 x \;\; A_H = a_2 x^2 = a_2$$
$$B(x) = b_0 + b_1 x + b_2 x^2 \therefore B_L = b_0 + b_1 x \;\; B_H = b_2 x^2 = b_2$$

El algoritmo Karatsuba-Ofman requiere del cálculo de los siguientes productos:

$$
\begin{aligned}
A_H B_H &= a_2 b_2 \\
A_L B_L &= (a_0 + a_1 x)(b_0 + b_1 x) \\
&= a_0 b_0 + (a_0 b_1 + a_1 b_0)\, x + a_1 b_1 x^2 \\
(A_H + A_L)(B_H + B_L) &= [(a_0 + a_2) + a_1 x]\,[(b_0 + b_2) + b_1 x] \\
&= (a_0 + a_2)\,(b_0 + b_2) + a_1 b_1 x^2 + \\
&\quad [(a_0 + a_1 + a_2)\,(b_0 + b_1 + b_2) + (a_0 + a_2)\,(b_0 + b_2) + a_1 b_1]\, x
\end{aligned}
\tag{2.8}
$$

En el ejemplo se puede ver que el término $a_1 b_1$ debe ser calculado para $A_L B_L$ y $(A_H + A_L)(B_H + B_L)$ esto podría hacerse utilizando un sólo cálculo para ambos productos. Lo anterior no ocurre cuando m es potencia de dos o par ya que no existen productos parciales comúnes.

En el algoritmo 2 se separan en dos partes cada uno de los operandos, utiliza el algoritmo medio no redundante 3 y a él mismo para realizar los productos correspondientes y finalmente une los resultados parciales generando el producto final. El valor de m_1 indica que los polinomios $a(x)_0^{m_1-1}$ y $b(x)_0^{m_1-1}$ son de grado $m-1$ y el valor de m_2 indica el número de coeficientes no redundantes que hay en los polinomios ADD_{a_x} y ADD_{b_x}.

En el algoritmo 3 se realizan al mismo tiempo dos productos en los cuales se evita realizar operaciones redundantes, para lo cual se utilizan los valores m_1 y m_2, dentro de este algoritmo se calculan los valores f y g, el primero indica el número de coeficientes que tienen los polinomios $a_1(x)_0^{m_1/2}$, $a_2(x)_0^{m_1/2}$, $ADD_{a_1(x)}$ y $ADD_{b_1(x)}$, el segundo indica que los polinomios $a_1(x)_{m_1/2}^{m_1-1}$, $a_2(x)_{m_1/2}^{m_1-1}$ tienen g coeficientes no redundantes.

2.2.8.3. Algoritmo Karatsuba-Ofman libre de traslape

En §2.2.8.1 (en la página 22) se explicó que la ventaja del algoritmo Karatsuba-Ofman se debe a que se dividen a los operandos en dos partes y se realizan operaciones parciales, esto es usando la técnica de divide y venceras, Zimmermann y Hanrot en [22] proponen dividir a las palabras de acuerdo a la paridad de los operandos y en [17] Fan *et al.* adaptan dicha técnica para realizar la multiplicación en $\mathbb{F}_2$ con el fin de lograr implementaciones en hardware más rápidas.

Para poder entender como funciona esta propuesta se definen dos parametros que son muy utilizados para medir el desempeño de la implementación de un multiplicador en hardware. La complejidad en área generalmente es representada por el número total de entradas a compuertas XOR and AND utilizadas. La complejidad en tiempo correspondiente está dada en términos del retraso que surge por la señales debido a estas compuertas. Donde, T_A se refiere al retraso que generan las compuertas AND y T_X es el retraso que generan las compuertas XOR.

El algoritmo 1 requiere un total de 3 retardos TX además del cálculo recursivo de las tres multiplicaciones parciales. La idea para esta propuesta consiste en dividir a los operandos de entrada de acuerdo a su paridad de la siguiente manera:

$$\begin{aligned} Ae(y) &= \sum_{i=0}^{m-1} a_{2i} y^i \\ Ao(y) &= \sum_{i=0}^{m-1} a_{2i+1} y^{i+1} \end{aligned}$$

de la misma manera se define $Be(y)$ y $Bo(y)$. Entonces los operandos A y B pueden escribirse: $A = A_e(y)+xA_o(y)$ y $B = B_e(y)+xB_o(y)$; es importante notar que estos polinomios son de grado menor que m por lo que las operaciones de multiplicación entre ellos se pueden realizar de manera recursiva. Siguiendo este método obtenemos esta fórmula:

$$\begin{aligned} A \cdot B &= (A_e(y) + xA_o(y))\,(B_e(y) + xB_o(y)) \\ &= \left\{A_e(y)B_e(y) + x^2 A_o(y)B_o(y)\right\} + x\,\{A_e(y)B_o(y) + A_o(y)B_e(y)\} \\ &= \{A_e(y)B_e(y) + yA_o(y)B_o(y)\} + \\ &\quad x\,\{[(A_e(y) + xA_o(y))\,(B_e(y) + xB_o(y))] + A_e(y)B_e(y) + A_o(y)B_o(y)\} \end{aligned}$$

Para una implementación en hardware de esta fórmula, multiplicar un polinomio por x o por $y = x^2$ es equivalente a realizar corrimientos y no se requiere de uso de compuertas.

Es fácil ver que en la expansión $\{A_e(y)B_e(y) + yA_o(y)B_o(y)\}$ solo hay términos con exponentes pares de x, esto porque $y = x^2$. En la expansión $x\,\{[(A_e(y) + xA_o(y))\,(B_e(y) + xB_o(y))] + A_e(y)B_e(y) + A_o(y)B_o(y)\}$ únicamente hay términos con exponentes impares de x.

De esta manera no existe ningun traslape al calcular la suma, por lo que no se requiere de compuertas. Por otra parte los terminos $[(A_e(y) + xA_o(y))\,(B_e(y) + xB_o(y))]$ y $[A_e(y)B_e(y) + A_o(y)B_o(y)]$ pueden ser calculados concurrentemente y la suma de estos operandos requiere un solo retardo T_x de la compuerta XOR. Por lo tanto se puede ver que el cálculo de AB utilizando este método requiere solamente un total de $2T_x$ además de el costo del calculo recursivo de las tres multiplicaciones parciales.

2.3. Curvas elípticas

Las curvas elípticas pueden ser definidas sobre los números reales, números complejos y en campos finitos. Se explicarán a continuación las curvas elípticas definidas sobre los números reales, ya que gracias a su representación geométrica se pueden comprender mejor sus propiedades. Sin embargo, desde el punto de vista criptográfico, sólo se emplean

curvas elípticas definidas sobre campos finitos. En esta tesis se estudian las curvas elípticas definidas sobre campos finitos binarios y campos finitos ternarios.

La ecuación (2.9) denota una curva elíptica de forma general conocida como forma de Weierstrass [21]:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.9}$$

En la figura 2.1 se presenta un ejemplo de curva elíptica de la forma $y^2 = x^3 + ax + b$ definida en $\mathbb{R}$ la cual es una forma simplificada de la ecuación de Weierstrass. Donde $a, b \in \mathbb{R}$, $b \neq 0$. Los puntos de coordenadas (x, y) que pertenecen a E son aquellos que satisfacen dicha ecuación, además del punto en el infinito, que se simboliza como ϑ.

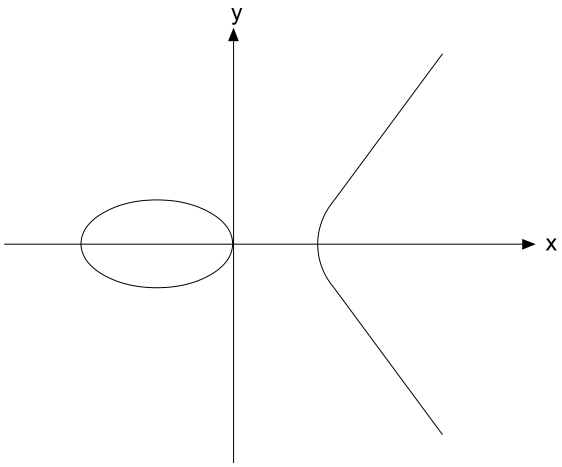

Figura 2.1: Curva elíptica de la forma $y^2 = x^3 + ax + b$ definida sobre $\mathbb{R}$.

El negativo de un punto P con coordenadas (x, y), es el reflejo de este punto sobre el eje x, es decir el punto con coordenadas $(x, -y)$. Es importante notar que el inverso de un punto P, que pertenece a E, también pertenece a E y se denota por $-P$ con coordenadas $(x, -y)$.

El grupo de puntos en una curva elíptica en $\mathbb{R}$ forma un grupo aditivo. En la Figura 2.2 se muestra la representación geométrica de la suma de dos puntos elípticos distintos. Sean P, Q dos puntos distintos sobre la curva E con coordenadas (x_1, y_1) y (x_2, y_2) respectivamente, donde $P \neq -Q$, se observa que la suma de puntos consiste en tomar la secante entre los puntos P y Q. La línea secante interseca a la curva en un punto $-R$ de coordenadas $(x_3, -y_3)$. El resultado de la suma de puntos $P + Q$ se obtiene reflejando el punto $-R$ sobre el eje x.

En el caso de que sumen los puntos P y $-P$, geométricamente se tiene que al trazar la secante entre ambos se obtiene una línea vertical, la cual no interseca la curva elíptica E. Por lo que la suma $P + (-P)$, no se puede obtener como en el caso anterior. Por esta razón

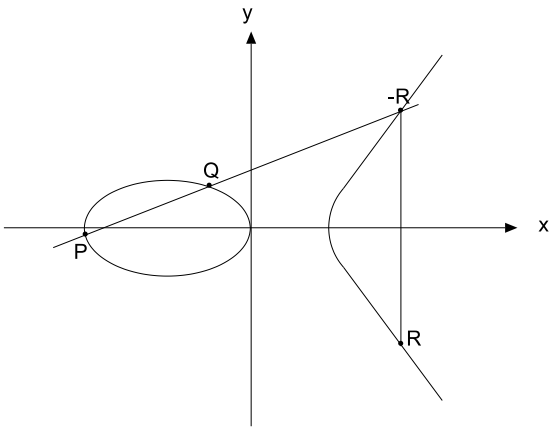

Figura 2.2: Representación gráfica de la suma de puntos en una curva elíptica definida sobre $\mathbb{R}$.

es que el grupo de curvas elípticas incluye el punto al infinito el cual se denota como ϑ. Por definición $P + (-P) = \vartheta$ y $P + \vartheta = P$, dentro del grupo aditivo de curvas elípticas. El elemento ϑ se conoce como elemento neutro aditivo.

Existe un tercer caso de suma de puntos elípticos, en el cual se suma el punto P a sí mismo, este tipo de suma también es conocido como doblado de un punto elíptico. La Figura 2.3 muestra que para calcular el doblado del punto P se debe traza una línea tangente a través del punto de coordenadas (x_1, y_1) de la curva elíptica, esta tangente interseca con la curva en el punto $-2P$ de coordenadas $(x_2, -y_2)$; el punto $2P$ con coordenadas (x_2, y_2) se obtiene reflejando el punto $-2P$ en el eje x.

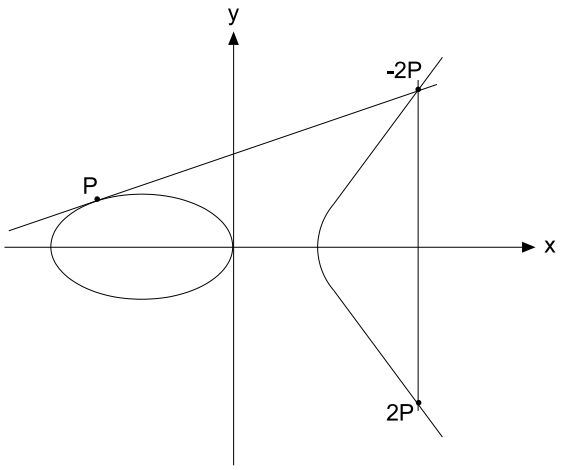

Figura 2.3: Representación gráfica del doblado de puntos en una curva elíptica definida sobre $\mathbb{R}$.

2.3.1. Curvas elípticas sobre campos finitos

Como se observó en §2.1.4.1 (en la página 13), los elementos que pertenecen al campo finito $\mathbb{F}_{2^m}$ se pueden representar como polinomios, por lo tanto pueden ser representados por cadenas de bits tomados de los coeficientes del polinomio, formalmente:

Sea E una curva elíptica sobre un campo finito binario $\mathbb{F}_{2^m}$ de la forma:

$$y^2 + xy = x^3 + ax^2 + b \tag{2.10}$$

donde $a, b \in \mathbb{F}_{2^m}$, $b \neq 0$; un punto P de coordenadas (x, y), pertenece a E si satisface la ecuación (2.10), además del punto en el infinito ϑ.

Al igual que en el caso de las curvas elípticas sobre el campo de los números reales, el grupo de curvas elípticas denotadas sobre un campo finito binario o ternario describen un grupo abeliano, el cual define la suma como la operación principal.

Sea una curva elíptica E definida sobre $\mathbb{F}_{q=p^n}$, el número de puntos que pertenecen a la curva se denota por $\#E(\mathbb{F}_q)$ y se conoce como el orden de la curva. Dichos puntos son todos aquellos que satisfacen la ecuación de la curva elíptica E más el elemento neutro aditivo ϑ mejor conocido como punto al infinito.

$$\#E(\mathbb{F}_q) = \left|\{(x, y) \in y^2 + xy - x^3 - ax^2 - b = 0\} \cup \{\vartheta\}\right|$$

Por otro lado, el orden de un punto se refiere al número de elementos que puede generar un punto P en la curva elíptica E. Sea P un punto de la curva $E(\mathbb{F}_q)$, el orden de P es el entero más pequeño k tal que $kP = \vartheta$.

En curvas elípticas es posible evaluar la multiplicación escalar kP la cual se obtiene mediante la suma de k veces el punto P, es decir:

$$Q = kP = \overbrace{P + P + \ldots + P}^{k \text{ veces}}.$$

El cálculo de dicha operación se lleva a cabo mediante suma y doblado de puntos.

2.3.2. Curvas elípticas supersingulares

Sea una curva elíptica E definida sobre $\mathbb{F}_{q=p^m}$ con p característica del campo y considerando que $t = q + 1 - \#E(\mathbb{F}_q)$. La curva E se denomina curva elíptica supersingular si y sólo si t es divisible por la característica del campo [52], es decir, $t \equiv 0 \bmod p$, condición que se cumple en el caso que $\#E(\mathbb{F}_{p^m}) \equiv 1 \bmod p$ en cuyo otro caso se dice que se trata de una curva elíptica ordinaria.

Cuando $p \geq 5$ se dice que E es supersingular sobre $\mathbb{F}_p$ si $t = 0$ cuyo caso implica que $\#E(\mathbb{F}_q) = p + 1$.

Debido a que el problema del logaritmo discreto en este tipo de curvas puede reducirse al problema del logaritmo discreto sobre un grupo multiplicativo de extensión $k-$ésima, $\mathbb{F}_q^k$ del campo $\mathbb{F}_q$ [31] se considera que se trata de curvas elípticas débiles, pero poseen la particularidad de que las operaciones entre los puntos que pertenecen a la curva son más eficientes que las que presentan las curvas ordinarias es por eso que se pueden utilizar para el cálculo de emparejamientos bilineales.

Como se mencionó anteriormente, desde el punto de vista criptográfico, sólo se emplean curvas elípticas definidas sobre campos finitos. En esta tesis estudiaremos curvas elípticas supersingulares definidas en $\mathbb{F}_{2^m}$, tal es el caso de la siguiente curva:

$$y^2 + y = x^3 + x + b, \tag{2.11}$$

donde $b \in 0, 1$, $k = 4$, es decir, el grado de seguridad es 4 y el número de puntos que pertenecen a la curva esta dado por $2m + 1 \pm 2^{(m+1)/2}$ conocido también como orden de la curva [2].

Capítulo 3

Multiplicadores Karatsuba-Ofman para $\mathbb{F}_{p^m}$ con arquitectura segmentada en hardware reconfigurable

En general las aplicaciones criptográficas requieren de implementaciones eficientes de la aritmética básica sobre campos finitos tales como suma, resta, multiplicación, división, exponenciación, cálculo de raíces, por mencionar algunas. De estas operaciones, la multiplicación polinomial es la que consume más recursos y tiempo de operación.

En este capítulo se describen de manera detallada los diseños e implementaciónes de algunas variantes del multiplicador Karatsuba-Ofman sobre los campos finitos $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{3^{97}}$ empleando distintos tipos de arquitectura como: serial, paralela y segmentada.

Todos los diseños se implementaron utilizando el lenguaje de descripción de hardware VHDL y se sintetizaron para una FPGA, específicamente para las familias Virtex II-Pro y Virtex 5 de Xilinx. Se ocuparon las herramientas de Xilinx ISE 8.1 y 9.2 para la descripción y Model SIM III 6.3c para la simulación.

3.1. Consideraciones de diseño e implementación

En esta tesis se diseñó e implementó el multiplicador más eficiente que cualquier otro reportado en la literatura abierta en $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{3^{97}}$, para lo cual se tomaron las siguientes consideraciones:

- En el primer nivel de la multiplicación se utilizó el algoritmo Karatsuba-Ofman, mientras que las multiplicaciones parciales son calculadas con su variante Karatsuba-Ofman no redundante.
- Es necesario el uso de la reducción modular para regresar el resultado de la multiplicación polinomial como un elemento que pertenece al campo finito.
- Se utilizó una representación en base polinomial para los elementos que pertenecen al campo finito. En $\mathbb{F}_{2^{239}}$ son representados como una cadena de bits, mientras que los

elementos en $\mathbb{F}_{3^{97}}$ son representados como una cadena de trits.

- Se realizaron pruebas del multiplicador polinomial truncando el algoritmo de Karatsuba-Ofman hasta obtener operandos de 16, 8 y 4 bits, siendo el último el que ofreció una mejor eficiencia.

- Se implementaron todos los algoritmos en la herramienta Maple para generar vectores de prueba y así poder validar los resultados obtenidos.

- Para ambas características se trabajó en los dispositivos Virtex V XC5VLX330 y Virtex II Pro XC2VP70 de Xilinx. La sintesis *place-and-route* se ejecutó utilizando las herramientas de diseño Xilinx ISE 9.2i y ModelSimSE 6.2c.

3.1.1. Rendimiento en FPGA

Cuando se realiza una implementación en un dispositivo FPGA existen varios criterios para medir su rendimiento, a continuación se describen de manera general las métricas utilizadas para la evaluación de un diseño en FPGA para así análizar los resultados obtenidos de los diseños descritos en este capítulo.

Área: se define como la cantidad de recursos que ocupa un diseño, se mide en *slices*, que son la medida basica. Dependiendo del diseño en ocasiones se pueden utilizar otros recursos internos como BRAMs, multiplicadores, etc., los cuales no ocupan slices, de esta existe un ahorro de recursos y es necesario reportarlo.

Retardo o tiempo de procesamiento: se refiere al tiempo que tarda un diseño en entregar el resultado en la salida.

Eficiencia: para hacer comparaciones con resultados que se presentan en la literatura abierta también medimos la eficiencia de los diseños, la cual esta dada por la siguiente ecuación:

$$Eficiencia = \frac{1}{área(\#Slices) * latencia}$$

Throughput: representa la cantidad de datos que se pueden procesar por segundos, en este caso está dado por $\frac{bits}{segundo}$.

3.2. Multiplicadores en $\mathbb{F}_{2^{239}}$

Como se describió en §2.2.8.1 (en la página 22) calcular la multiplicación polinomial haciendo uso del algoritmo Karatsuba-Ofman que consiste en dividir a cada uno de los operandos en dos palabras, ver Algoritmo 1 (en la página 24), se dividió a los operandos de la siguiente manera:

A		B	
A_H	$A\,(238:120)$	B_H	$(238:120)$
A_L	$A\,(119:0)$	B_L	$(119:0)$

Las Figuras 3.1 y 3.2 representan el modelo de cajas para el algoritmo Karatsuba-Ofman 1 y su variante libre de traslape descrito en §2.2.8.3 respectivamente, recordemos que son necesarias tres multiplicaciones parciales las cuales se pueden obtener utilizando un multiplicador en $\mathbb{F}_{2^{120}}$, sin embargo, 120 tampoco es una potencia de dos por lo que se decidió utilizar la técnica del algoritmo Karatsuba-Ofman no redundante descrita en el Algoritmo 2, por las ventajas que ofrece en estos casos.

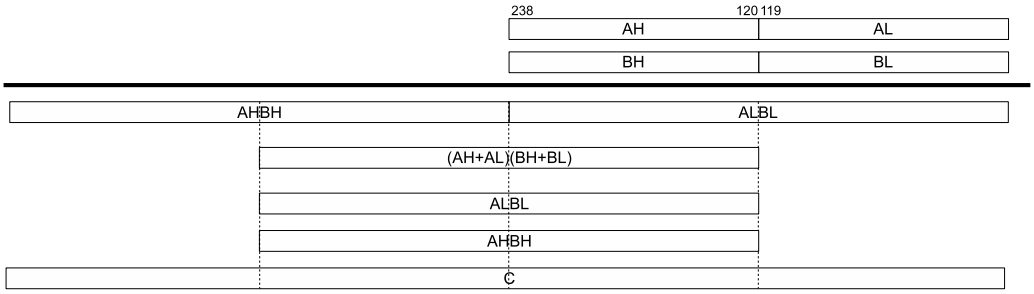

Figura 3.1: Algoritmo Karatsuba-Ofman

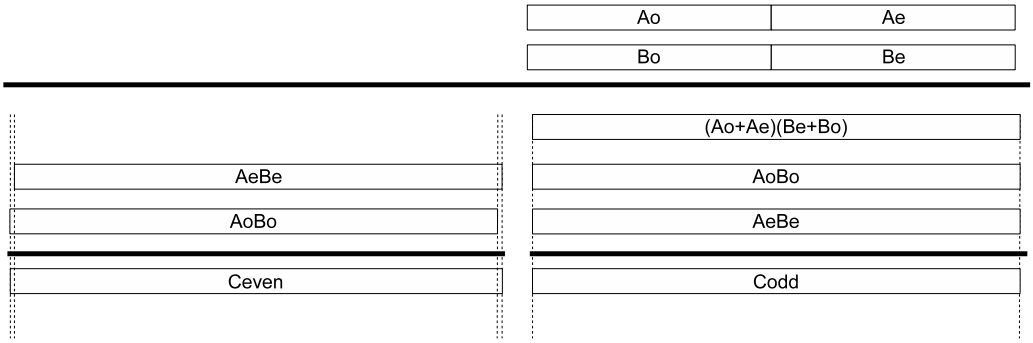

Figura 3.2: Algoritmo Karatsuba-Ofman libre de traslape

En la Figura 3.3 se muestra la arquitectura general del diseño del multiplicador en $\mathbb{F}_{2^{120}}$, de lado izquierdo se representa al módulo Mul64_56 el cual realiza dos multiplicaciones entre operandos que cuentan con 56 bits en común evitando con esto realizar cálculos redundantes y de lado derecho se representa el módulo Mul56 que realiza solamente una multiplicación.

Al construir el multiplicador en $\mathbb{F}_{2^{239}}$ se obtiene como resultado un elemento fuera del campo de aproximádamente 478 bits por esta razón es necesaria una reducción modular, para lograrlo en esta implementación se utilizó el trinomio irreducible: $x^{239} + x^{81} + 1$ donde sólo es necesario el uso de compuertas XOR.

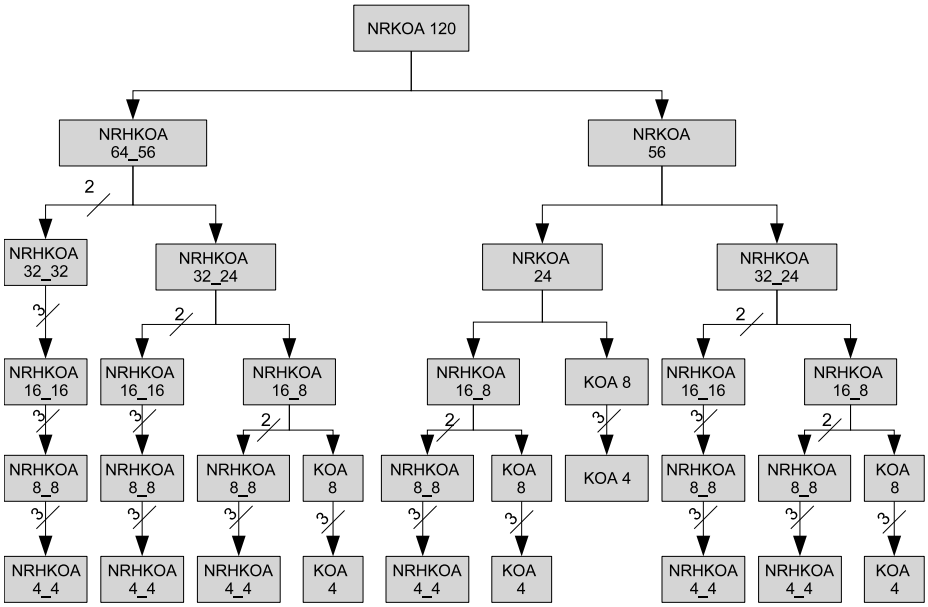

Figura 3.3: Arquitectura general del mutiplicador en $\mathbb{F}_{2^{120}}$

Como se ha mencionado el algoritmo Karatsuba-Ofman requiere del cálculo de tres multiplicaciones parciales. Por razones de desempeño y eficiencia en implementaciones en hardware en este caso se puede diseñar segmentando las tareas (ya que no existen dependencias entre los datos), haciendo uso de uno, dos o tres multiplicadores, de tal manera que se reduzca la trayectoría crítica a diferencia de las arquitecturas completamente paralelas donde la trayectoría crítica es cara.

Considerando lo anterior y con el propósito de realizar un análisis de desempeño, se diseñaron e implementaron las tres opciones empleando las técnicas descritas en §2.2.8.1 (en la página 22) y §2.2.8.3 (en la página 27).

3.2.1. Arquitectura segmentada

La segmentación (en inglés *pipelining*) es un método por el cual se consigue aumentar el rendimiento de algún sistema. En hardware los cálculos deben ser sincronizados con el reloj cada cierto tiempo para que la ruta crítica (retardo computacional entre dos registros de reloj o tramo con más carga) se reduzca. La ruta crítica es en realidad la frecuencia máxima de trabajo alcanzada por el circuito. A mayor ruta crítica (retraso entre registros) menor es la frecuencia máxima de trabajo y a menor ruta crítica mayor frecuencia de trabajo.

Repartir o segmentar el cálculo de alguna operación hace que esa frecuencia sea la óptima a costa de más área para el almacenamiento de los datos necesarios y de la latencia (en ciclos de reloj * tiempo) en la salida del resultado equivalente al número de segmentaciones realizados. La ventaja primordial de esta arquitectura es que una vez el

pipe está lleno los resultados posteriores vienen uno tras otro cada ciclo de reloj y sin latencia extra por estar encadenados dentro del mismo *pipe*. Con lo anterior se busca la frecuencia máxima del diseño.

A continuación se explican a detalle los diseños con arquitectura segmentada para esta tesis.

3.2.1.1. Arquitectura segmentada de una etapa

Las Figuras 3.4 y 3.5 muestran la arquitectura general y el diagrama de tiempos asociados a la arquitectura segmentada de una etapa, es decir, haciendo uso de un sólo multiplicador. Como se puede ver en la Figura 3.4 nuestra estructura se dividió en dos bloques: el primero de ellos realiza la multiplicación, mientras que en el otro se lleva a cabo el traslape y la reducción modular.

Se decidió utilizar al multiplicador en el primer ciclo de reloj para evaluar el producto parcial $(A_i^H + A_i^L)(B_i^L + B_i^H)$, en el segundo ciclo de reloj se evalua $(A_i^L + B_i^L)$ y en el tercer ciclo de reloj se obtiene el tercer resultado parcial dado por $(A_i^H + B_i^H)$. Se requiere un ciclo de reloj adicional para realizar los traslapes entre los resultados parciales obtenidos, ver Figura 3.1, además de que es necesario otro ciclo de reloj para realizar la reducción modular. En la Figura 3.5 se puede apreciar la ventaja que ofrece nuestra arquitectura segmentada ya que es posible reutilizar al multiplicador durante estos dos últimos ciclos y de esta manera a partir del cuarto ciclo de reloj se puede calcular un producto cada tres ciclos de reloj. Por lo anterior cada unidad de tiempo representa 3 ciclos de reloj.

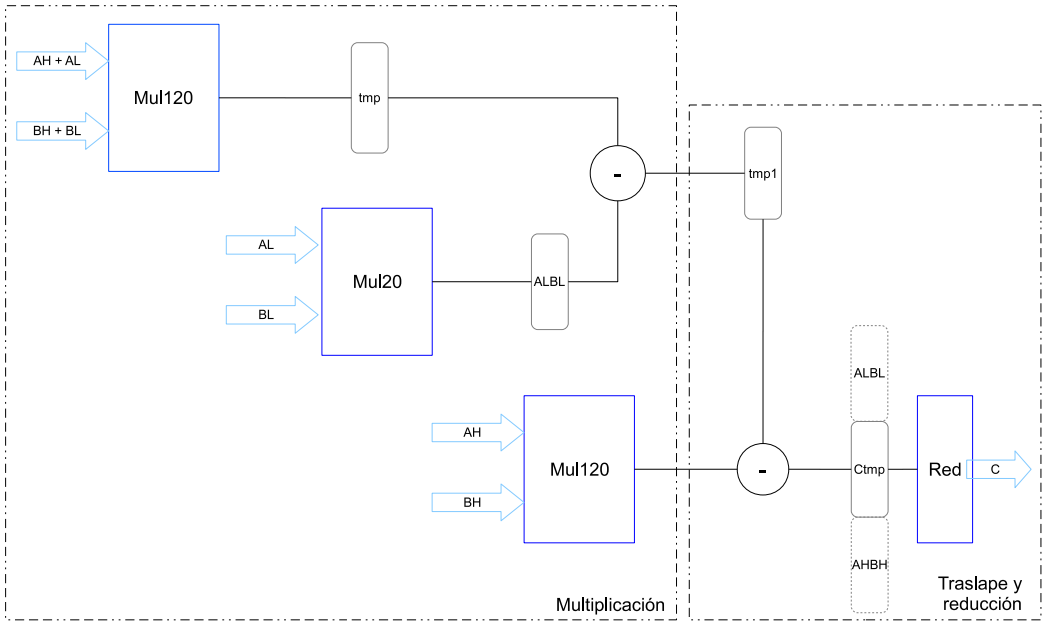

Figura 3.4: Diagrama a bloques del multiplicador en $\mathbb{F}_{2^{239}}$: arquitectura segmentada de una etapa

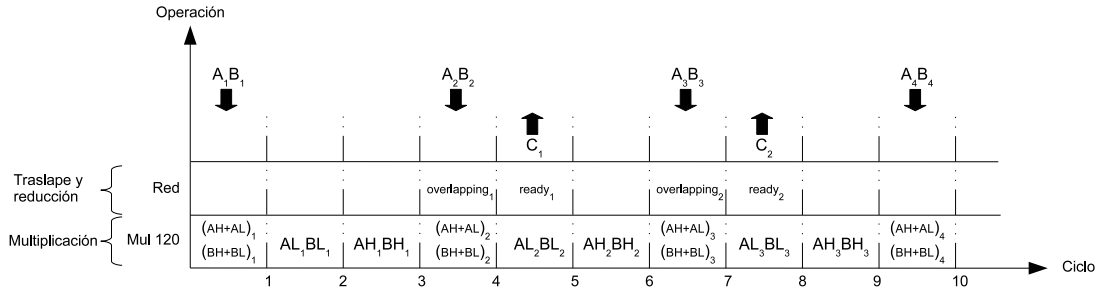

Figura 3.5: Diagrama de tiempos para un multiplicador en $\mathbb{F}_{2^{239}}$: arquitectura segmentada de una etapa

3.2.1.2. Arquitectura segmentada de dos etapas

Siguiendo la misma estrategia de la arquitectura segmentada de un estado, esta vez se emplean dos módulos de multiplicación, ver Figura 3.6. Durante los tres primeros ciclos de reloj el primer multiplicador evalúa los productos parciales $(A_i^H + A_i^L)(B_i^L + B_i^H)$, $(A_i^H + B_i^H)$ y $(A_{i+1}^H + A_{i+1}^L)(B_{i+1}^L + B_{i+1}^H)$, mientras que el segundo multiplicador se utiliza para calcular los productos $(A_i^L + B_i^L)$, $(A_{i+1}^H + B_{i+1}^H)$ y $(A_{i+1}^L + B_{i+1}^L)$.

Los resultados de los multiplicadores son direccionados a dos bloques de demultiplexores que almacenan el valor en registros temporales ó directamente se suma dicha salida del multiplicador con otros productos previamente obtenidos, esto depende del ciclo en el que se encuentre. Se puede observar que los productos $(A_i^L + B_i^L)$ y $(A_i^H + B_i^H)$ son necesarios como entradas al módulo de traslape, esto se controla haciendo uso de los multiplexores 1 y 3; por otro lado el multiplexor número 2 se encarga de colectar el producto parcial $(A_i^H + A_i^L)(B_i^L + B_i^H)$, el cual puede provenir de dos bloques sumadores diferentes. Las señales de control S_0, S_1 y S_2 se proporcionan por la unidad de control que organizan el flujo de datos como se muestra en el diagrama de tiempos de la Figura 3.7.

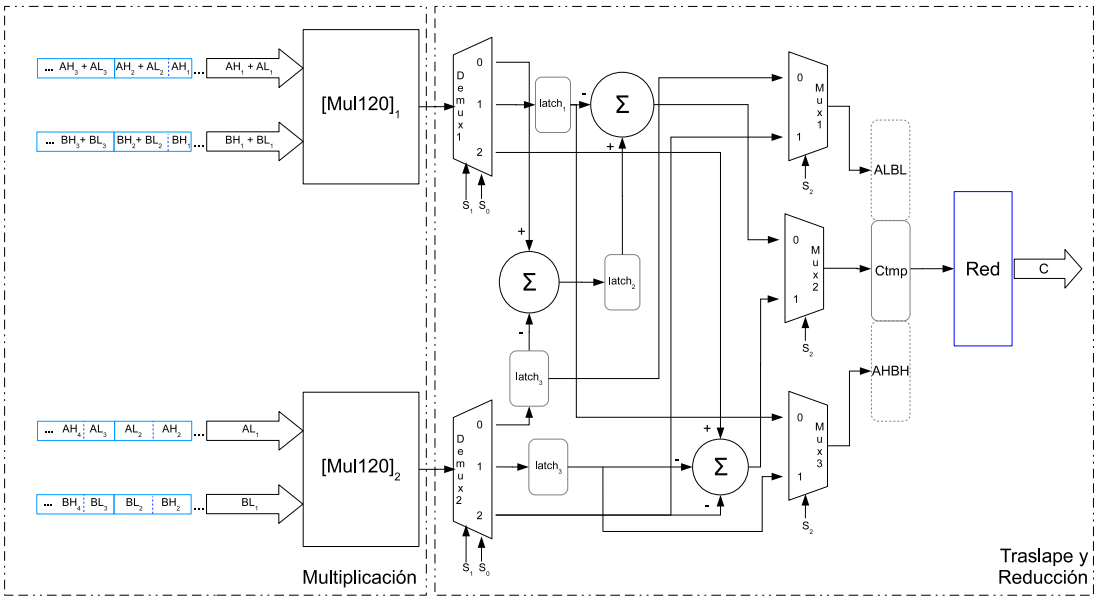

Figura 3.6: Diagrama a bloques del multiplicador en $\mathbb{F}_{2^{239}}$: arquitectura segmentada de dos etapas

El flujo de datos de entrada y salida de esta arquitectura se explica a continuación:

- A la entrada en los primeros dos ciclos de reloj se recibe a los operandos A_i, B_i, A_{i+1} y B_{i+1}, es necesario esperar un ciclo de reloj para poder recibir $A_{i+2}, B_{i+2}, A_{i+3}$ y B_{i+3} en los siguientes dos ciclos de reloj y así sucesivamente.

- A la salida los dos primeros resultados están listos en el tercero y cuarto ciclo de reloj respectivamente, correspondientes a los operandos recibidos en el primero y segundo ciclo de reloj; los siguientes dos resultados se presentan en el sexto y séptimo ciclo de reloj y así sucesivamente.

De acuerdo a esta arquitectura se realizan dos multiplicaciones por cada tres ciclos de reloj, es decir, la unidad de tiempo representa $\frac{2}{3}$ ciclos de reloj.

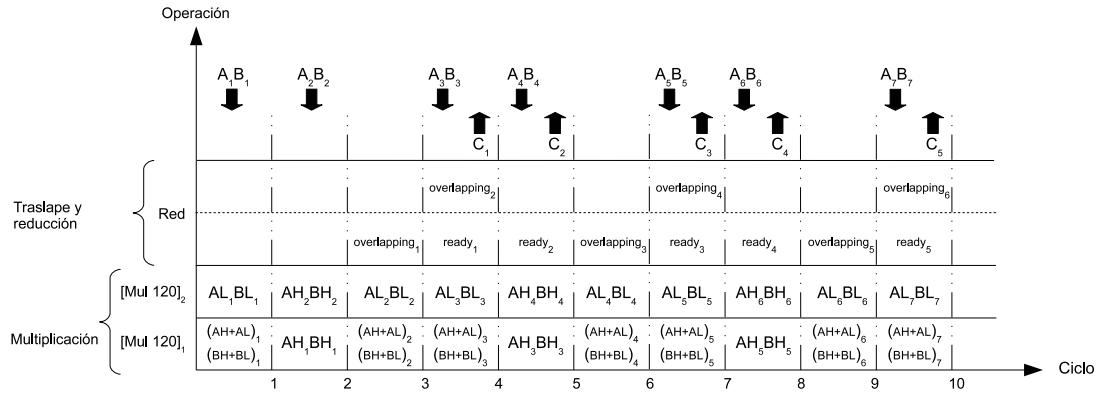

Figura 3.7: Diagrama de tiempos para un multiplicador en $\mathbb{F}_{2^{239}}$: arquitectura segmentada de dos etapas

3.2.1.3. Arquitectura segmentada de tres etapas

En esta arquitectura se utilizan tres multiplicadores, los cuales evalúan los productos parciales en un sólo ciclo de reloj, como se muestra en la Figura 3.8. En el segundo ciclo de reloj se realiza el traslape entre los productos parciales, en este momento los tres multiplicadores pueden ser reutilizados. Es importante considerar que los valores de $A_i^L B_i^L$ y $A_i^H B_i^H$ deben de guardarse en registros temporales ya que son necesarios como entrada para el módulo de reducción que se lleva a cabo en el tercer ciclo de reloj.

Como se muestra en la Figura 3.9, dada la organización de los dos componentes básicos (multiplicador y reductor) el procesamiento consiste en introducir los operandos de manera consecutiva; el primer resultado tardará 2 ciclos; ya que se aprovechan los multiplicadores al mismo tiempo que se realiza el traslape se logra que los resultados sean consecutivos. Por lo anterior cada unidad de tiempo representa 1 ciclo de reloj.

La Figura 3.9 muestra el diagrama de tiempos correspondiente a la arquitectura segmentada de tres etapas, donde el comportamiento de entrada y sálida se da de la siguiente forma:

- A la entrada se recibe el par de operandos a multiplicar en cada ciclo de reloj.

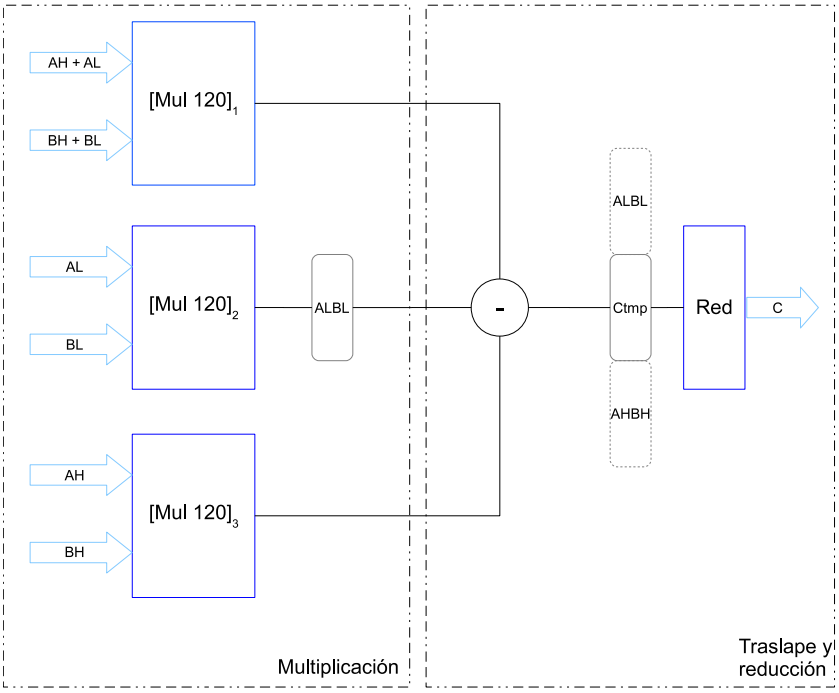

Figura 3.8: Diagrama a bloques del multiplicador en $\mathbb{F}_{2^{239}}$: arquitectura segmentada de tres etapas

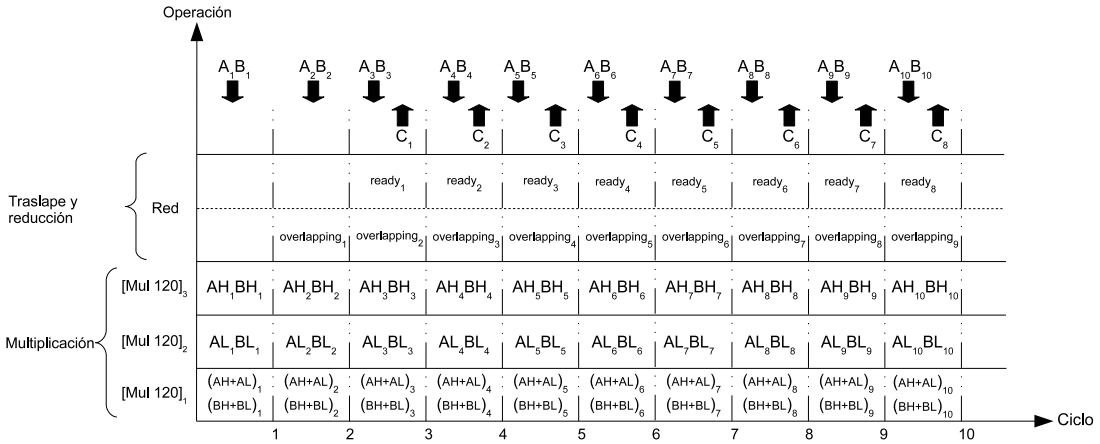

Figura 3.9: Diagrama de tiempos para un multiplicador en $\mathbb{F}_{2^{239}}$: arquitectura segmentada de tres etapas

- Partiendo del segundo ciclo de reloj los resultados se obtienen a cada ciclo de reloj.

De acuerdo a esta arquitectura se realiza una multiplicación por cada ciclo de reloj, es decir, la unidad de tiempo representa 1 ciclo de reloj, este comportamiento es similar a una arquitectura completamente paralela, que se describe en la §2.2.8 (en la página 20).

Para todas las arquitecturas anteriormente descritas se implementó la variante de Karatsuba-Ofman libre de traslape donde se divide a los operandos de acuerdo a la paridad de sus bits, véase §2.2.8.3 (en la página 27). Finalmente el producto se obtuvo como se muestra en la Figura 3.2, donde se puede apreciar que existe ahorro en el módulo de traslape y es necesario reacomodar a los bits antes de realizar la reducción modular.

3.2.2. Resultados del multiplicador en $\mathbb{F}_{2^{239}}$

En la tabla 3.1 se presentan los resultados obtenidos para la implementación en $\mathbb{F}_{2^{239}}$ de acuerdo a los diseños que se explicaron en §3.2. La columna de tiempo se refiere a un sólo ciclo de reloj y la columna de latencia esta dada por el número de ciclos multiplicado por el tiempo que se refiere al tiempo total que se requiere para obtener el resultado.

El trabajo de Estibals [16] es el más reciente, sus implementaciones presentan mayor frecuencia que nuestros diseños, sin embargo, los nuestros son mucho más rápidos y optimizamos el área de todos los diseños. Obtuvimos los multiplicadores más eficientes que cualquier otro reportado en la literatura abierta.

Tabla 3.1: Resultados de la implementación en $\mathbb{F}_{2^{239}}$

Multiplicador	Ciclos	Tiempo [ns]	Latencia [ns]	Frecuencia [MHz]	Área [Slices]	Eficiencia	Throughput [Mbits/seg]
Una etapa [H⊥L]	3	9.24	27.72	108.14	1,865	19,343	8,622
Dos etapas [H⊥L]	$\frac{3}{2}$	9.31	13.96	107.41	3,631	19,728	17,120
Tres etapas [H⊥L]	1	9.96	9.96	100.38	4,751	21,133	23,996
Una etapa [E_O]	3	8.95	26.85	111.71	2,034	18,310	8,901
Dos etapas [E_O]	$\frac{3}{2}$	9.89	14.84	101.05	3,659	18,411	16,105
Tres etapas [E_O]	1	9.61	9.61	104.05	4,651	22,373	24,870
Estibals [16]	5	7.53	37.67	133.00	10,897	2,436	6,345
	5	6.66	33.34	150.00	12,490	2,401	7,169

3.3. Multiplicadores en $\mathbb{F}_{3^{97}}$

En esta sección se presenta el diseño y la implementación de un multiplicador polinomial ternario basado en la combinación de diferentes variantes de esquemas de Karatsuba-Ofman recientemente publicados [17] y [10]. Se implementó el multiplicador en arquitectura paralela así como en arquitectura segmentada de k etapas con $k = 1, 2, 3, 4$ donde cada etapa esta compuesta por un multiplicador de 49 trits.

3.3.1. Arquitectura paralela

Se implementó el algoritmo Karatsuba-Ofman para el campo finito $\mathbb{F}_{3^{97}}$ siguiendo la técnica descrita en §2.2.8.2 (en la página 24), cada elemento en este campo se representa como una palabra de 97 trits la cual ocupa 194 bits de memoria; la arquitectura utilizada para este diseño se muestra en la Figura 3.10.

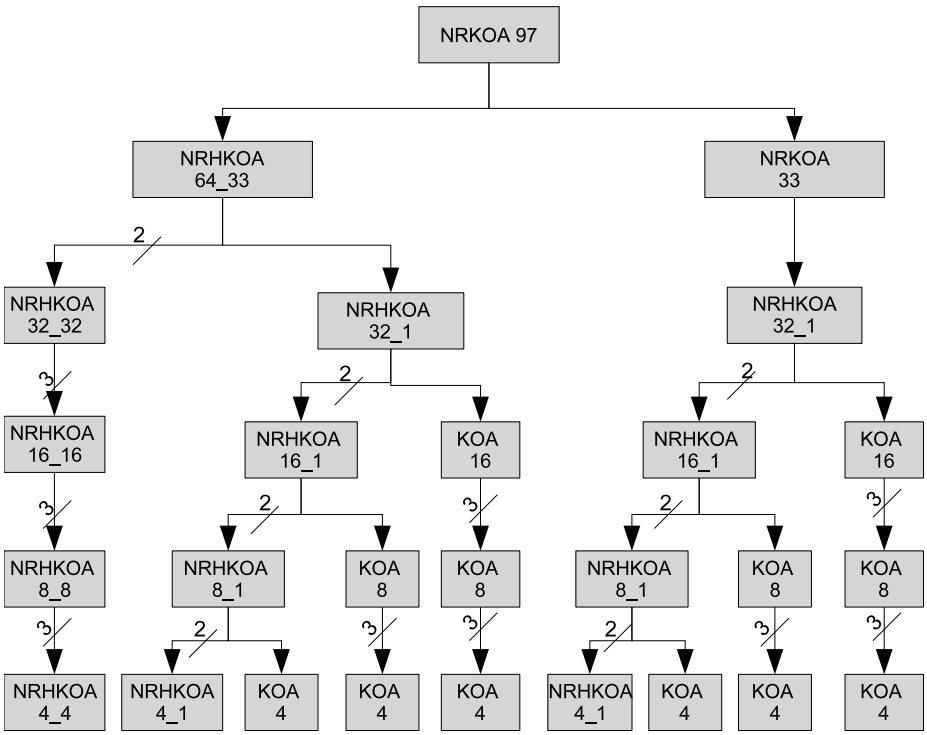

Figura 3.10: Arquitectura del multiplicador en $\mathbb{F}_{3^{97}}$

Finalmente se implementó la reducción modular haciendo uso del trinomio irreducible $x^{97} + x^{12} + 1$ tal y como se muestra en la Figura 3.11.

Debido a que las pruebas realizadas a esta arquitectura mostraban que la eficiencia de este diseño no era lo suficientemente competitiva se decidió continuar con el diseño de arquitecturas segmentadas.

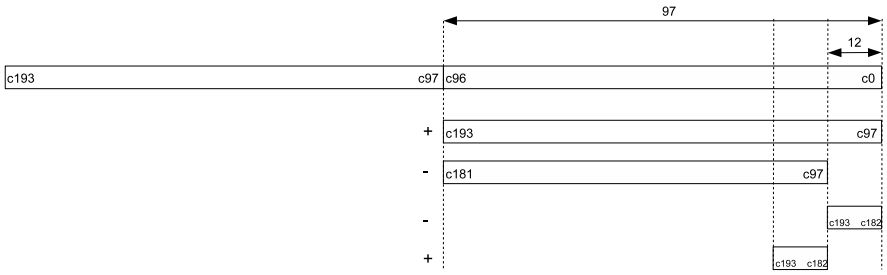

Figura 3.11: Reducción modular para $\mathbb{F}_{3^{97}}$

3.3.2. Arquitectura segmentada

Siguiendo la idea de § 3.2.1 se dividió a los operandos en palabras de 49 trits; dado que 49 tampoco es una potencia de dos se implementó este multiplicador siguiendo la técnica del algoritmo Karatsuba-Ofman no redundante [10].

La Figura 3.12 muestra la arquitectura general para este diseño, donde se presentan dos módulos principales: de lado izquierdo el Mul32_17 el cual realiza dos multiplicaciones y de lado derecho el Mul17 que evalua una sola multiplicación.

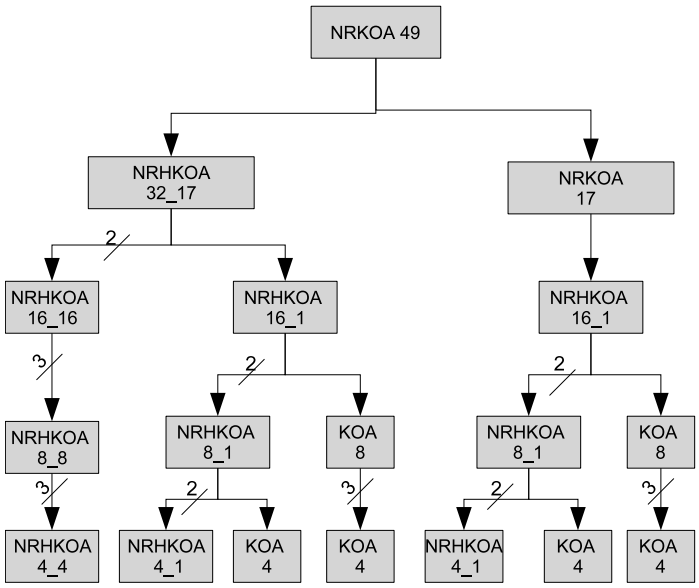

Figura 3.12: Arquitectura para el multiplicador en $\mathbb{F}_{3^{49}}$

Con el fin de ilustrar el ahorro que existe al utilizar la técnica no redundante, la Figura 3.13 muestra un Mul16_1 desenrrollado hasta los multiplicadores de 4 trits, que es donde se deja de utilizar la técnica Karatsuba-Ofman y se emplea un algoritmo de multiplicación clásico.

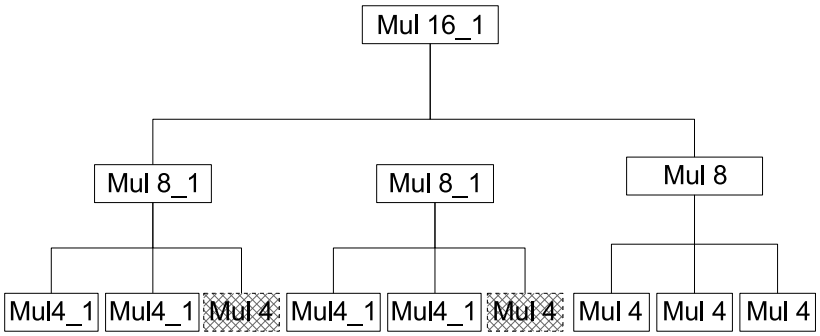

Figura 3.13: Arquitectura para un mul 16_1

Se puede construir el multiplicador en $\mathbb{F}_{3^{97}}$ haciendo uso de uno, dos, tres y hasta cuatro multiplicadores en $\mathbb{F}_{3^{49}}$; esto es posible ya que los recursos hardware son suficientes. A continuación se describe a detalle cada uno de estos diseños.

3.3.2.1. Arquitectura segmentada de una etapa

Las Figuras 3.14 y 3.15 muestran la arquitectura general y el diagrama de tiempos asociados a la arquitectura segmentada de una etapa. Como se puede ver en la Figura 3.14 esta arquitectura es análoga a la descrita en § 3.2.1.3 tomando en cuenta las siguientes diferencias:

- La característica del campo finito es 3 y su extensión es $\mathbb{F}_{3^{97}}$.
- El módulo de multiplicación es de 49 trits, es decir, para evaluar los productos parciales se reciben operandos de 49 trits.
- En este campo finito la reducción implica operaciones de suma y resta.

El comportamiento del diagrama de tiempos es análogo al descrito en la Figura 3.5.

3.3.2.2. Arquitectura segmentada de dos etapas

En la Figura 3.16 se presenta la arquitectura segmentada de dos etapas la cual es análoga a la mostrada en la Figura 3.6. Se siguió la misma estrategia de diseño cambiando únicamente el tamaño de las palabras de entrada a los módulos y teniendo en cuenta las consideraciones que se describen en §3.3.2.1. El diagrama de tiempos se muestra en la Figura 3.17 que es análoga a la Figura 3.7 previamente descrita.

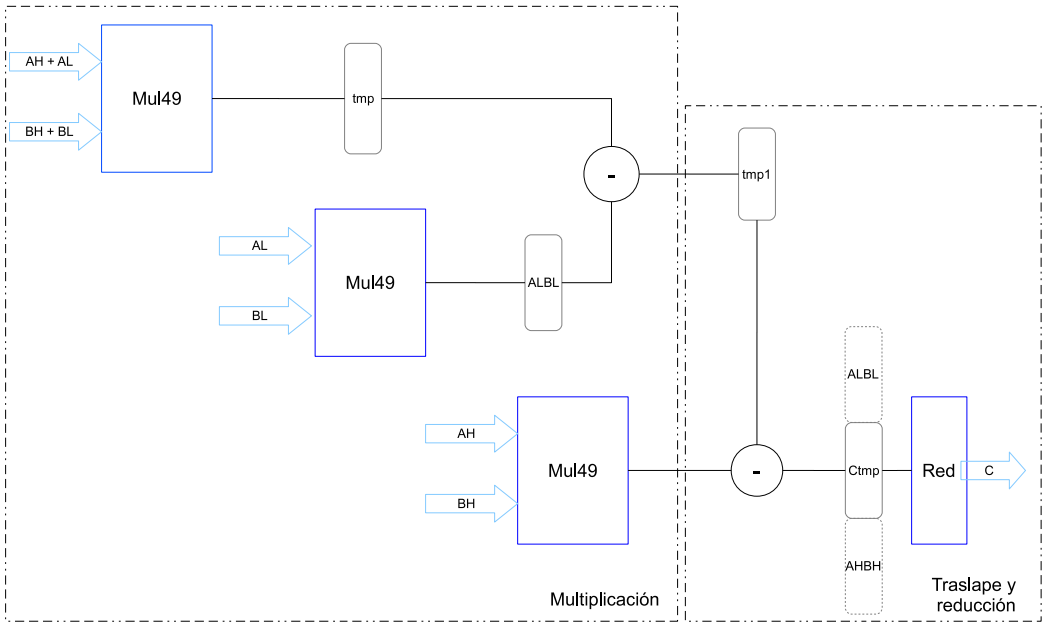

Figura 3.14: Diagrama a bloques del multiplicador con arquitectura segmentada de una etapa

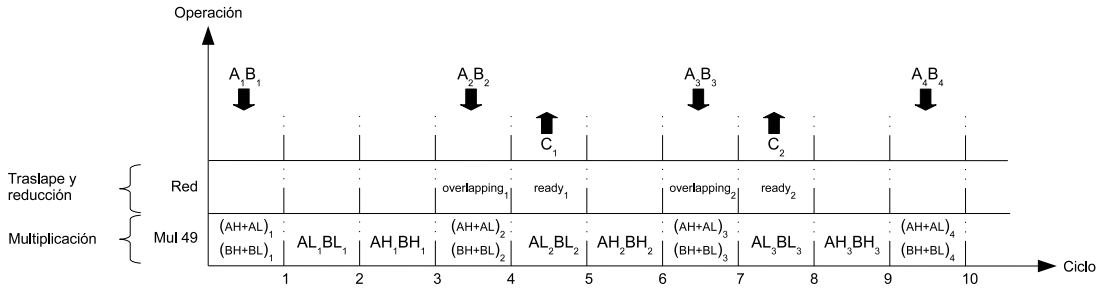

Figura 3.15: Diagrama de tiempos para un multiplicador en $\mathbb{F}_{3^{97}}$ con arquitectura segmentada de una etapa

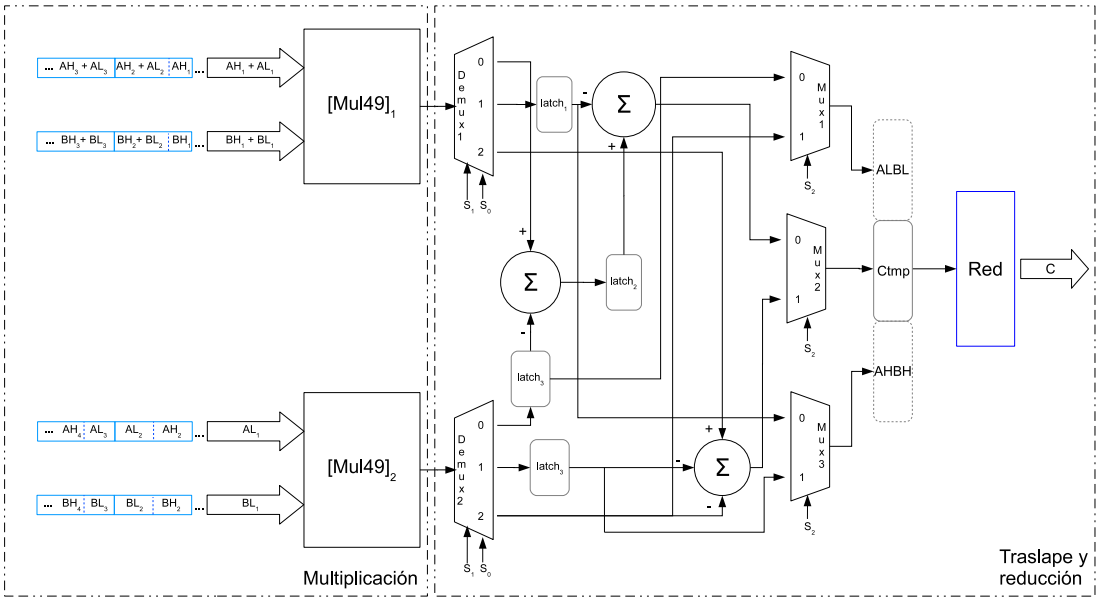

Figura 3.16: Diagrama a bloques del multiplicador en $\mathbb{F}_{3^{97}}$ con arquitectura segmentada de dos etapas

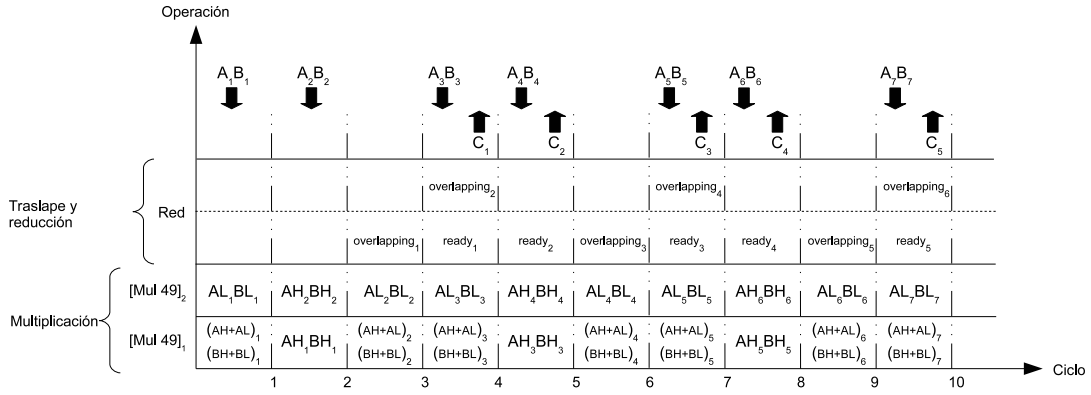

Figura 3.17: Diagrama de tiempos para un multiplicador en $\mathbb{F}_{3^{97}}$ con arquitectura segmentada de dos etapas

3.3.2.3. Arquitectura segmentada de tres etapas

Las Figuras 3.18 y 3.19 muestran la arquitectura y el diagrama de tiempos para esta arquitectura segmentada de tres etapas. Dichas Figuras son análogas a las descritas en §3.2.1.3, donde ya fueron explicadas a detalle.

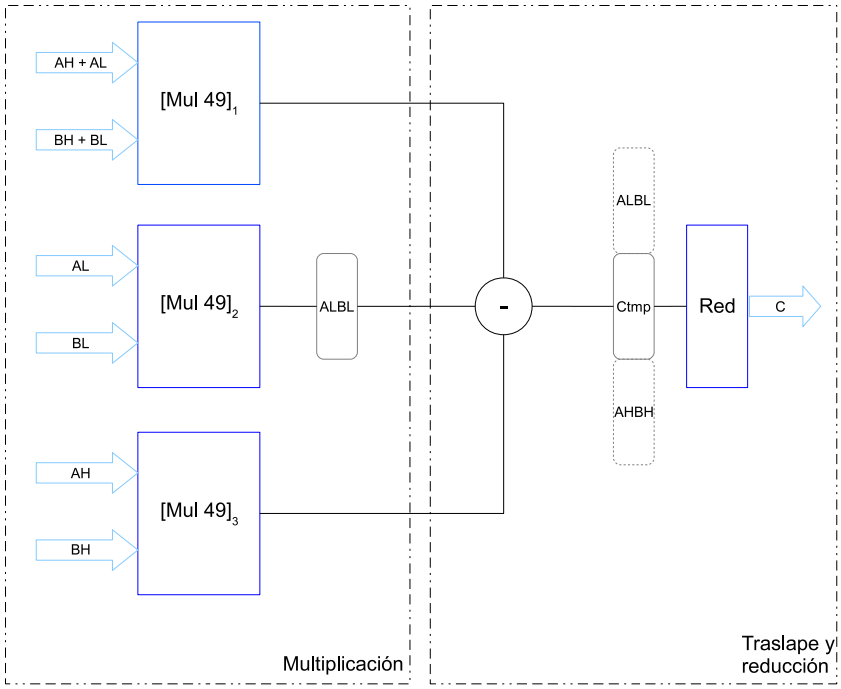

Figura 3.18: Diagrama a bloques del multiplicador en $\mathbb{F}_{3^{97}}$: arquitectura segmentada de tres etapas

3.3.2.4. Arquitectura segmentada de cuatro etapas

Para caracteristica tres se implementó la arquitectura segmentada de cuatro etapas ya que los recursos de hardware fueron suficientes. La Figura 3.20 muestra el diagrama de tiempos para esta arquitectura donde se puede ver que a partir del segundo ciclo de reloj es posible calcular cuatro multiplicaciones parciales cada tres ciclos de reloj, lo que implica que éste diseño ofrece una gran capacidad de cálculo considerando que puede computar más de un producto parcial en cada ciclo de reloj.

En este diseño la unidad de tiempo representa $\frac{4}{3}$ ciclo de reloj.

Para todas las arquitecturas anteriormente descritas también se implementó la variante de Karatsuba-Ofman libre de traslape donde se divide a los operandos de acuerdo a la paridad de sus bits, ver § 2.2.8.3, finalmente el producto se obtuvo como se muestra en la Figura 3.2, donde se puede apreciar que existe ahorro en el módulo de traslape y es necesario reacomodar a los bits antes de realizar la reducción modular.

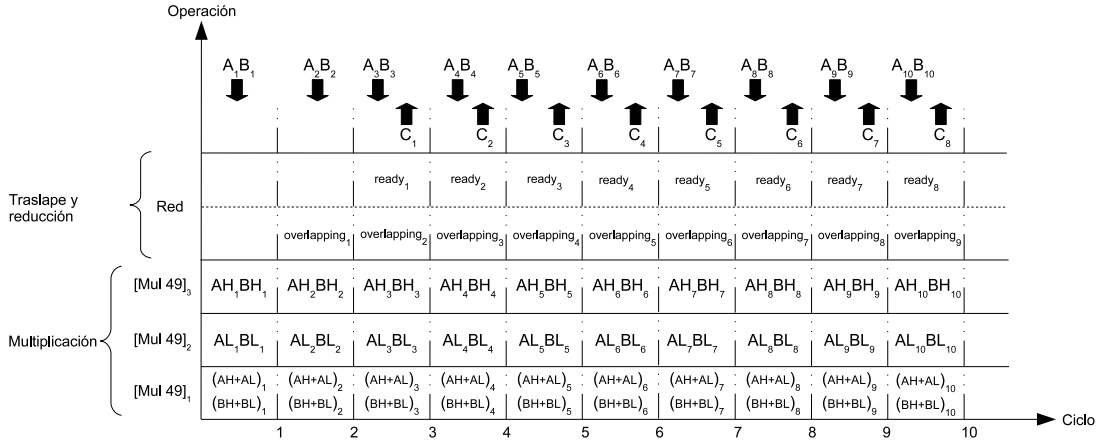

Figura 3.19: Diagrama de tiempos para un multiplicador en $\mathbb{F}_{3^{97}}$: arquitectura segmentada de tres etapas

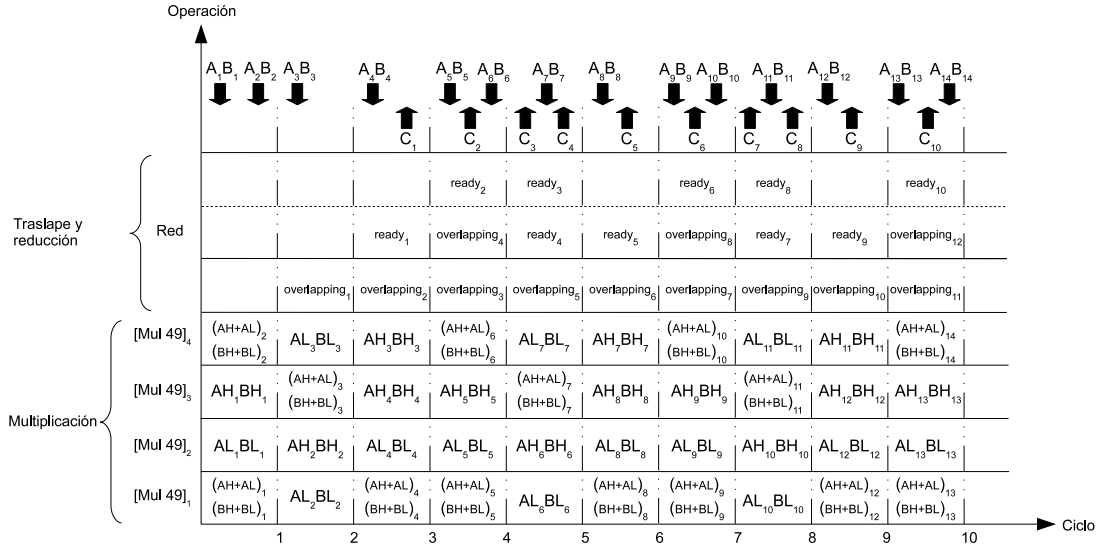

Figura 3.20: Diagrama de tiempos para un multiplicador en $\mathbb{F}_{3^{97}}$: arquitectura segmentada de cuatro etapas

Tabla 3.2: Resultados de la implementación en $\mathbb{F}_{3^{97}}$ con arquitectura completamente paralela

Arquitectura del multiplicador	Ciclos	Tiempo (ns)	Latencia (ns)	Area (Slices)	Eficiencia
Completamente paralela	1	29.49	29.49	5281	6419

Al utilizar esta técnica tanto en característica dos como en característica tres, se observó que el algoritmo Karatsuba-Ofman no redundante es eficiente únicamente cuando se divide a los operandos en: palabra alta y palabra baja; ya que de esta manera los bits que tienen en común quedan sólo en la palabra alta y se emplean los dos algoritmos propuestos por los autores en [10], sin embargo, al dividirlos de acuerdo a su paridad existen bits en común tanto en las palabras pares como en las palabras impares, así que se tendrían que realizar módulos específicos para poder evitar las multiplicaciones reduntantes siguiendo la técnica propuesta en [10] pero modificando los algoritmos 2 y 3. Por lo anterior se decidio dividir a los operandos de acuerdo a su paridad sólamente en el nivel más alto de nuestros multiplicadores.

De acuerdo a las implementaciones realizadas, la técnica Karatsuba-Ofman libre de traslape sólo presentó ventajas para los diseños en característica tres ya que el ahorro de una compuerta XOR en el traslape que los autores mencionan en [17] de manera teórica no se refleja de manera práctica dada la organización de los bloques lógicos configurables (CLBs) de los dispositivos utilizados, ver Apendice A.

3.3.3. Resultados del multiplicador en $\mathbb{F}_{3^{97}}$

Los resultados de la implementación del multiplicador completamente paralelo se presentan en la tabla 3.2.

En la tabla 3.3 se muestran los resultados de la implementación de los multiplicadores con arquitectura segmentada. Al notar que los resultados de los diseños reportados recientemente en la literatura abierta estan implementados en un dispositivo Virtex II Pro de Xilinx se decidio implementar nuestros diseños en un XC2VP70 el cual pertenece a la misma familia. Éste material sirvió para un artículo publicado como poster en el congreso *Cryptographic Hardware and Embedded Systems* (CHES 2008), véase [11]. Así como en $\mathbb{F}_{2^{239}}$ en $\mathbb{F}_{3^{97}}$ obtuvimos los multiplicadores más rápidos y los más eficientes que cualquier otro reportado en la literatura abierta.

Tabla 3.3: Resultados de la implementación en $\mathbb{F}_{3^{97}}$ con arquitectura segmentada

Multiplicador	Ciclos	Tiempo [ns]	Latencia [ns]	Area [Slices]	Eficiencia	Throughput [Mtrits/seg]
Paralela	1	41.59	41.59	4,915	4,891	2,332
Segmentada: una etapa [H⊥L]	3	44.63	133.89	2,518	9,318	725
Segmentada: dos etapas [H⊥L]	$\frac{3}{2}$	23.15	34.72	4,171	10,353	2,794
Segmentada: tres etapas [H⊥L]	1	14.71	14.71	5,328	12,753	6,594
Segmentada: cuatro etapas [H⊥L]	$\frac{3}{4}$	12.04	9.03	8,634	9,616	10,743
Segmentada: un etapa [E_O]	3	35.81	107.43	2,337	11,947	903
Segmentada: dos etapas [E_O]	$\frac{3}{2}$	21.32	31.98	4,163	11,262	3,033
Segmentada: tres etapas [E_O]	1	13.22	13.22	4,895	15,442	8,798
Segmentada: cuatro etapas [E_O]	$\frac{3}{4}$	11.46	8.58	7,740	11,267	11,305
Beuchat *et al.* [4]	33	6.71	221.43	700	6,450	438
Shokrollahi *et al.* [49]	97	3.33	323.01	327	9,458	300
	14	9.00	126.00	2,954	2,684	770
	7	13.90	97.30	4,006	2,568	997
Ronan *et al.* [43]	7	16.23	113.61	3,737	2,356	854
Grabher *et al.* [20]	28	6.67	186.76	946	5,665	519
Kerings *et al.* [27]	25	34.12	853.00	1,821	644	114
Bertoni *et al.* [3]	7	10.60	74.20	3,561	3,787	1,307
Estibals [16]	7	7.71	53.94	10,316	1,797	1,798
	7	7.99	55.91	10,403	1,719	1,735

Capítulo 4

Caso de estudio: exponenciación final en $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{2^{313}}$

Los emparejamientos bilineales fueron introducidos en criptografía por Menezes [31] y Frey [18] como herramienta para criptoanálisis; posteriormente Joux [25], Sakai [45] y Mitsunari [37] dan comienzo a un sin número de propuestas de protocolos basados en emparejamientos debido a que presentan las propiedades constructivas de los emparejamientos.

Estudiando de manera general el trabajo realizado por Beuchat *et al.*en [4], [7] y especialmente en [5] donde presentan que el cálculo del emparejamiento requiere del cómputo eficiente del ciclo principal y el de una operación adicional denominada exponenciación final y que dichas operaciones requieren del cómputo eficiente de muchas multiplicaciones se decidió utilizar los multiplicadores descritos en el capítulo anterior para la construcción del módulo de exponenciación final en $\mathbb{F}_{2^m}$.

En este capítulo se describe la construcción de la aritmética en torres de campos ya que es necesaria para realizar cualquiera de las operaciones que conforman al emparejamiento, se presenta la definición de los emparejamientos así como algunos algoritmos para evaluar el ciclo principal y se detalla el algoritmo de la exponenciación final. Finalmente se describe el diseño y la implementación de ésta operación en hardware reconfigurable.

De manera análoga a los diseños mostrados en §3.2 el diseño de la exponenciación final se implementó utilizando el lenguaje de descripción de hardware VHDL y se sintetizó para una FPGA; específicamente para las familias Virtex II-Pro y Virtex 5 de Xilinx. Se ocuparon las herramientas de Xilinx ISE 8.1 y 9.2 para la descripción y Model SIM III 6.3c para la simulación.

4.1. Aritmética en torres de campos $\mathbb{F}_{2^{km}}$

Como se mencionó, para lograr el cálculo de la exponenciación final se requiere de la construcción de la aritmética en torre de campos para $\mathbb{F}_{2^{2m}}$ y $\mathbb{F}_{2^{4m}}$.

4.1.1. Aritmética en torre de campos $\mathbb{F}_{2^{2m}}$

A continuación se definen las operaciones de suma, multiplicación, elevar al cuadrado e inversión en $\mathbb{F}_{2^{2m}}$, donde los elementos se representan utilizando la base $\{1, s\}$.

$$U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}; \quad u_0, u_1 \in \mathbb{F}_{2^m}$$

4.1.1.1. Suma

Es importante recordar que en los campos binarios la operación de suma es equivalente a la operación de resta, en el campo $\mathbb{F}_{2^{2m}}$ esta operación se realiza coeficiente a coeficiente utilizando las operaciones descritas en §2.2 para $\mathbb{F}_{2^m}$.

Sean $U, V \in \mathbb{F}_{2^{2m}}$ donde:

$$U = u_0 + u_1 s$$
$$V = v_0 + v_1 s,$$

la operación de suma se define así:

$$\begin{aligned} U + V &= (u_0 + u_1 s) + (v_0 + v_1 s) \\ &= (u_0 + v_0) + (u_1 + v_1)s \end{aligned}$$

4.1.1.2. Multiplicación polinomial

Se puede realizar una multiplicación en $\mathbb{F}_{2^{2m}}$ utilizando la técnica de Karatsuba-Ofman, lo cual implica el cálculo de tres multiplicaciones y cuatro sumas en $\mathbb{F}_{2^m}$.

Sean $U, V \in \mathbb{F}_{2^{2m}}$ y considerando que $s^2 = s + 1$ la multiplicación se define así:

$$\begin{aligned} UV &= (u_0 + u_1 s)(v_0 + v_1 s) \\ &= u_0 v_0 + (u_0 v_1 + u_1 v_0)s + u_1 v_1 s^2 \\ &= u_0 v_0 + u_1 v_1 + (u_0 v_1 + u_1 v_0 + u_1 v_1)s \\ &= u_0 v_0 + u_1 v_1 + ((u_0 + u_1)(v_0 + v_1) + u_0 v_0)s \end{aligned} \tag{4.1}$$

4.1.1.3. Elevar al cuadrado

Elevar al cuadrado un elemento de $\mathbb{F}_{2^{2m}}$ se hace de manera directa. Sea $U \in \mathbb{F}_{2^{2m}}$ y considerando que $s^2 = s + 1$ la operación de elevar al cuadrado se define como:

$$\begin{aligned} U^2 &= (u_0 + u_1 s)^2 \\ &= u_0^2 + u_1^2 s^2 \\ &= (u_0^2 + u_1^2) + u_1^2 s \end{aligned}$$

4.1.1.4. Inversión

Evaluar la exponenciación final requiere de una inversión en $\mathbb{F}_{2^{2m}}$, véase algoritmo 4. Sea $V = v_0 + v_1 s \in \mathbb{F}_{2^{2m}}$ el elemento inverso multiplicativo de $U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}$, $U \neq 0$, donde u_0, u_1, v_0 y $v_1 \in \mathbb{F}_{2^m}$. Dado que $UV = 1$ se tiene que:

$$\begin{aligned} u_0 v_0 + u_1 v_1 &= 1 \\ u_0 v_1 + u_1 v_0 + u_1 v_1 &= 0 \end{aligned}$$

La solución a este sistema de ecuaciones esta dado por:

$$\begin{aligned} v_0 &= w^{-1}(u_0 + u_1) \\ v_1 &= w^{-1} u_1, \end{aligned}$$

donde $w = u_0^2 + (u_0 + u_1)u_1 \in \mathbb{F}_{2^m}$. De esta manera, la inversión en $\mathbb{F}_{2^{2m}}$ implica tres multiplicaciones, dos sumas, una elevación al cuadrado y una inversión en $\mathbb{F}_{2^m}$.

Algoritmo 4: Inversión en $\mathbb{F}_{2^{2m}}$

Entrada: $U = u_0 + u_1 s \in \mathbb{F}_{2^{2m}}$, $U \neq 0$.
Salida: $V = U^{-1} = v_0 + v_1 s \in \mathbb{F}_{2^{2m}}$

1 $a_0 \leftarrow u_0 + u_1$;
2 $m_0 \leftarrow u_0^2$; $m_1 \leftarrow a_0 u_1$;
3 $a_1 \leftarrow m_0 + m_1$;
4 $i_0 \leftarrow a_1^{-1}$;
5 $v_0 \leftarrow a_0 i_0$;
6 $v_1 \leftarrow u_1 i_0$;
7 **return** $v_0 + v_1 s$

4.1.2. Aritmética en torre de campos $\mathbb{F}_{2^{4m}}$

Una vez que se cuenta con la aritmética en torre de campo $\mathbb{F}_{2^{2m}}$ se requiere de la aritmetica en torre de campo $\mathbb{F}_{2^{4m}}$ para poder realizar el cálculo de la exponenciación final.

A continuación se definen las operaciones se suma, multiplicación y elevar al cuadrado $\mathbb{F}_{2^{4m}}$, donde los elementos se representan utilizando la base $\{1, s, t, st\}$.

$$W = w_0 + w_1 s + w_2 t + w_3 st \in \mathbb{F}_{2^{2m}}; \quad w_0, w_1, w2, w3 \in \mathbb{F}_{2^m}$$

Se definen las siguientes propiedades: $s^2 = s + 1$ y $t^2 = t + s$.

4.1.2.1. Suma

De manera análoga al campo $\mathbb{F}_{2^{2m}}$ esta operación se realiza coeficiente a coeficiente utilizando las operaciones descritas en §2.2 para $\mathbb{F}_{2^m}$.

Sean $U, V \in \mathbb{F}_{2^{4m}}$ donde:

$$U = u_0 + u_1 s + u_2 t + u_3 st$$

$$V = v_0 + v_1 s + v_2 t + v_3 st,$$

la operación de suma se define así:

$$\begin{aligned} U + V &= (u_0 + u_1 s + u_2 t + u_3 st) + (v_0 + v_1 s + v_2 t + v_3 st) \\ &= (u_0 + v_0) + (u_1 + v_1)s + (u_2 + v_2)t + (u_3 + v_3)st \end{aligned}$$

4.1.2.2. Multiplicación polinomial

El algoritmo 5 muestra la multiplicación en $\mathbb{F}_{2^{4m}}$, se realiza siguiendo la técnica de Karatsuba-Ofman, ver alg. 1 para el cual se requiere del cálculo de nueve multiplicaciones y veinte sumas en $\mathbb{F}_{2^m}$.

Algoritmo 5: Multiplicación en $F_{2^{4m}}$

Entrada: $U = u_0 + u_1 s + u_2 t + u_3 st, V = v_0 + v_1 s + v_2 t + v_3 st \in \mathbb{F}_{2^{4m}}$
Salida: $W = UV \in \mathbb{F}_{2^{4m}}$

1 $a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow v_0 + v_1$;
2 $a_2 \leftarrow u_0 + u_2$; $a_3 \leftarrow v_0 + v_2$;
3 $a_4 \leftarrow u_1 + u_3$; $a_5 \leftarrow v_1 + v_3$;
4 $a_6 \leftarrow u_2 + u_3$; $a_7 \leftarrow v_2 + v_3$;
5 $a_8 \leftarrow a_0 + a_6$; $a_9 \leftarrow a_1 + a_7$;
6 $m_0 \leftarrow u_0 v_0$; $m_1 \leftarrow u_1 v_1$; $m_2 \leftarrow u_2 v_2$; $m_3 \leftarrow u_3 v_3$;
7 $m_4 \leftarrow a_0 a_1$; $m_5 \leftarrow a_2 a_3$; $m_6 \leftarrow a_4 a_5$; $m_7 \leftarrow a_6 a_7$; $m_8 \leftarrow a_8 a_9$;
8 $a_{10} \leftarrow m_0 + m_1$; $a_{11} \leftarrow m_0 + m_4$;
9 $w_0 \leftarrow a_{10} + m_2 + m_7$;
10 $w_1 \leftarrow a_{11} + m_3 + m_7$;
11 $w_2 \leftarrow a_{10} + m_5 + m_6$;
12 $w_3 \leftarrow a_{11} + m_5 + m_8$;
13 **return** $w_0 + w_1 s + w_2 t + w_3 st$

4.1.2.3. Elevar al cuadrado

Otra operación importante es la de elevar al cuadrado en $\mathbb{F}_{2^{4m}}$. Sea $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$. $V = U^2$ esta dado por $U^2 = u_0^2 + u_1^2 s^2 + u_2^2 t^2 + u_3^2 s^2 t^2$. Dado que $s^2 = s+1, t^2 = t+s$ y $s^2 t^2 = 1 + t + st$ se obtienen los siguientes coeficientes para V:

$$\begin{aligned} v_0 &= u_0^2 + u_1^2 + u_3^2 \\ v_1 &= u_1^2 + u_2^2 \\ v_2 &= u_2^2 + u_3^2 \\ v_3 &= u_3^2 \end{aligned}$$

De modo tal que con cuatro elevaciones al cuadrado y cuatro sumas en $\mathbb{F}_{2^m}$ es posible calcular $V = U^2$.

4.1.2.4. Exponenciación: U^{2^m+1}

Finalmente se requiere del cálculo de U^{2^m+1} en $\mathbb{F}_{2^{4m}}$. Sea $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$, considerando que $s^{2^m} = s + 1$ y $t^{2^m} = t + s + \alpha + 1$, se obtiene:

$$U^{2^m} \begin{cases} (u_0 + u_1 + u_3) + (u_1 + u_2)s + (u_2 + u_3)t + u_3 st & si\ \alpha = 1, \\ (u_0 + u_1 + u_2) + (u_1 + u_2 + u_3)s + (u_2 + u_3)t + u_3 st & si\ \alpha = 0. \end{cases}$$

El algoritmo 6 muestra una solución para calcular U^{2^m+1}, la cual requiere del cálculo de cinco multiplicaciones, dos elevaciones al cuadrado y nueve sumas en $\mathbb{F}_{2^m}$.

Algoritmo 6: Cálculo de U^{2^m+1} en $\mathbb{F}_{2^{4m}}$

Entrada: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}_{2^{4m}}$
Salida: $V = U^{2^m+1}$

$a_0 \leftarrow u_0 + u_1$; $a_1 \leftarrow u_2 + u_3$;
$m_0 \leftarrow a_0 a_1$; $m_1 \leftarrow u_0 u_1$; $m_2 \leftarrow u_0 u_3$;
$m_3 \leftarrow u_1 + u_2$; $m_4 \leftarrow u_2 + u_3$;
$s_0 \leftarrow a_0^2$; $s_1 \leftarrow a_1^2$;
$v_3 \leftarrow m_4 + s_1$;
$v_2 \leftarrow m_2 + m_3$;
if $\alpha = 1$ **then**
    $v_1 \leftarrow v_3 + m_0 + m_2$;
    $v_0 \leftarrow v_2 + m_1 + s_0$;
    $v_2 \leftarrow v_2 + v_3$;
else
    $v_1 \leftarrow v_3 + m_0 + m_3$;
    $v_0 \leftarrow m_0 + m_1 + m_2 + s_0$;
end
return $v_0 + v_1 s + v_2 t + v_3 st$

4.2. Emparejamientos bilineales

4.2.1. Definición

Sea n un número primo, $G_1 = \langle P \rangle$ un grupo aditivo de orden n con identidad ∞ y sea G_T un grupo multiplicativo de orden n con identidad 1.

Un *emparejamiento bilineal* sobre (G_1, G_T) es un mapeo

$$\hat{e} = G_1 \times G_1 \rightarrow G_T$$

que satisface las siguientes condiciones [32]:

Bilinealidad Para todo $R, S, T \in G_1$,

$$\begin{aligned} \hat{e}(R+S,T) &= \hat{e}(R,T)\hat{e}(S,T)y \\ \hat{e}(R,S+T) &= \hat{e}(R,S)\hat{e}(R,T). \end{aligned}$$

No degeneración $\hat{e}(P,P) \neq 1$.

Calculabilidad Existe un algoritmo eficiente para calcular $\hat{e}(P,Q)$ para todo $P, Q \in G_1$. Debido a que el cálculo de emparejamientos bilineales implica el cómputo de operaciones como suma, multiplicación, exponenciación e inversión entre otras es necesario implementar dichas operaciones de manera eficiente, es decir, buscar un equilibrio entre área y latencia.

Las siguientes propiedades de los emparejamientos bilineales pueden ser fácilmente verificadas. La propiedad número cinco es otra manera de definir la propiedad de no degeneración. Para todo $S, T \in G_1$:

1. $\hat{e}(S,\infty) = 1$ y $\hat{e}(\infty,S) = 1$.
2. $\hat{e}(S,-T) = \hat{e}(-S,T) = \hat{e}(S,T)^{-1}$.
3. $\hat{e}(aS,bT) = \hat{e}(S,T)^{ab}$ para todo $a, b \in \mathbb{Z}$.
4. $\hat{e}(S,T) = \hat{e}(T,S)$
5. Si $\hat{e}(S,R) = 1$ para todo $R \in G_1$, entonces $S = \infty$

Una consecuencia de la propiedad de bilinealidad es que el problema del logaritmo discreto en G_1 puede ser reducido de manera eficiente al problema del logaritmo discreto en G_T. Si (P,Q) es una instancia del problema del logaritmo discreto en G_1 donde $Q = xP$ entonces $\hat{e}(P,Q) = \hat{e}(P,xP) = \hat{e}(P,P)^x$ de esta manera $\log_P Q = \log_g h$ donde $g = \hat{e}(P,P)$ y $h = \hat{e}(P,Q)$ son elementos de G_T.

4.2.2. Algoritmos de emparejamiento

Miller describió el primer algoritmo iterativo para calcular el emparejamiento de Tate y el emparejamiento de Weil en [35], [36]; de los cuales en la práctica el primero de ellos es mucho más eficiente al tratar con curvas elípticas que proveen niveles de seguridad comunes [4].

En el año 2002 Barreto *et al.* [2] y Galbraith *et al.* [19] muestran algunas mejoras significativas al algoritmo de Miller y es desde entonces que se ha puesto especial atención a las curvas elípticas supersingulares. Un año después Duursma y Lee en [14] describieron una fórmula similar para característica tres. En el año 2004 Barreto *et al.* en [27] presentaron el método η_T el cual, entre otras características, reduce el ciclo del algoritmo de Miller siendo este último método el de interés en esta tesis.

Recientemente en [5] Beuchat *et al.* presentan un estudio del algoritmo modificado de Tate en característica dos y tres. Los algoritmos 7 y 8 surgen a partir de mejoras que realizaron al algoritmo propuesto por Barreto *et al.* en [27].

Considerando la ecuación (2.11) que denota a una curva elíptica supersingular, se define $\delta = b$ si $m \equiv 1, 7 \bmod 8$ y $\delta = 1 - b$ en otro caso. Y $\nu = (-1)^{\delta}$ [2]. Es importante notar que ambos algoritmos deben devolver el valor de F^M donde:

$$M = (2^{2m} - 1)(2^m + 1) + \nu(1 - 2^{2m})2^{\frac{m+1}{2}}, \tag{4.2}$$

esta operación es conocida como exponenciación final la cual se explicará a detalle ya que fue el caso de estudio en esta tesis.

Algoritmo 7: Cálculo del emparejamiento η_T en característica dos, con raíces cuadradas [5]

Entrada: $P, Q \in \mathbb{F}_{2^m}[l]$, M definido en ecuación (4.2)
Salida: $\eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$

$y_P \leftarrow y_P + \overline{\delta}$;
$u \leftarrow x_P + \alpha$; $v \leftarrow x_Q + \alpha$;
$g_0 \leftarrow uv + y_P + y_Q + \beta$;
$g_1 \leftarrow u + x_Q$; $g_2 \leftarrow v + x_P^2$;
$G \leftarrow g_0 + g_1 s + t$;
$L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$;
$F \leftarrow LG$;
for $j \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
    $x_P \leftarrow \sqrt{x_P}$; $y_P \leftarrow \sqrt{y_P}$; $x_Q \leftarrow x_Q^2$; $y_Q \leftarrow y_Q^2$;
    $u \leftarrow x_P + \alpha$; $v \leftarrow x_Q + \alpha$;
    $g_0 \leftarrow uv + y_P + y_Q + \beta$;
    $g_1 \leftarrow u + x_Q$;
    $G \leftarrow g_0 + g_1 s + t$;
    $F \leftarrow FG$;
end
return F^M;

Algoritmo 8: Cálculo del emparejamiento η_T en característica dos, sin raíces cuadradas [5]

Entrada: $P, Q \in \mathbb{F}_{2^m}[l]$, M definido en ecuación (4.2)
Salida: $\eta_T(P, Q) \in \mathbb{F}^*_{2^{4m}}$

$y_P \leftarrow y_P + \overline{\delta}$;
$x_P \leftarrow x_P^2$; $y_P \leftarrow y_P^2$;
$y_P \leftarrow y_P + b$; $u \leftarrow x_P + 1$;
$g_1 \leftarrow u + x_Q$; $g_0 \leftarrow x_P x_Q + y_P + y_Q + g_1$;
$x_Q \leftarrow x_Q + 1$;
$g_2 \leftarrow x_P^2 + x_Q$;
$G \leftarrow g_0 + g_1 s + t$;
$L \leftarrow (g_0 + g_2) + (g_1 + 1)s + t$;
$F \leftarrow LG$; **for** $j \leftarrow 1$ **to** $\frac{m-1}{2}$ **do**
    $F \leftarrow F^2$;
    $x_Q \leftarrow x_Q^4$; $y_Q \leftarrow y_Q^4$;
    $x_Q \leftarrow x_Q + 1$; $y_Q \leftarrow y_Q + x_Q$;
    $g_0 \leftarrow u x_Q + y_P + y_Q$;
    $g_1 \leftarrow x_P + x_Q$;
    $G \leftarrow g_0 + g_1 s + t$;
    $F \leftarrow FG$;
end
return F^M

4.2.3. Algoritmo de exponenciación final en $\mathbb{F}_{2^m}$

Se han propuesto diversos algoritmos para realizar el cálculo de la exponenciación final, véase ecuación 4.2, los cuales buscan disminuir el número de inversiones necesarias ya que esta operación es muy costosa.

- Ronan *et al.* en [44] suponen que $\nu = 1$, desarrollaron las diferentes potencias y agruparon las inversiones con lo que su algoritmo de exponenciación final requiere sólo una inversión en $\mathbb{F}_{2^{4m}}$.

- Shu *et al.* en [50] notaron que al elevar el emparejamiento η_T a la potencia $2^{2m} - 1$ solamente requiere una inversión en $\mathbb{F}_{2^{2m}}$. Cuando $\nu = -1$ la segunda parte de la exponenciación final consiste en elevar dicho resultado intermedio a la potencia $2^m + 1 + 2^{\frac{m+1}{2}}$.

- Beuchat *et al.* en [5] mostraron que la exponenciación final del algoritmo de emparejamiento η_T en característica dos siempre requiere de una sóla inversión en $\mathbb{F}_{2^{2m}}$.

Este último se explicará a detalle ya que se utilizó en las implementaciones presentadas en esta tesis.

Considerando la ecuación 4.2 se puede ver que:

$$\eta_T(P,Q)^M = \left(\eta_T(P,Q)^{2^{2m}+1}\right)^{2^m+1} \left(\eta_T(P,Q)^{\nu 1-2^{2m}}\right)^{2^{\frac{m+1}{2}}}$$

Sea $U = \eta_T(P,Q) \in \mathbb{F}^*_{2^{4m}}$. Considerando $U = U_0 + U_1 t$, donde $U_0, U_1 \in \mathbb{F}_{2^{2m}}$ y $t^{2^{2m}} = t+1$ se obtiene que $U^{2^{2m}} = U_0 + U_1 + U_1 t$. Por lo tanto:

$$\begin{aligned} U^{2^{2m}-1} &= \frac{U_0 + U_1 + U_1 t}{U_0 + U_1 t} = \frac{(U_0 + U_1 + U_1 t)^2}{(U_0 + U_1 t)(U_0 + U_1 + U_1 t)} \\ &= \frac{U_0^2 + U_1^2 + U_1^2 s + U_1^2 t}{U_0^2 + U_0 U_1 + U_1^2 s} \\ U^{1-2^{2m}} &= \frac{U_0 + U_1 t}{U_0 + U_1 + U_1 t} = \frac{U_0^2 + U_1^2 s + U_1^2 t}{U_0^2 + U_0 U_1 + U_1^2 s}, \end{aligned}$$

donde $U_0^2 + U_0 U_1 + U_1^2 s \in \mathbb{F}_{2^{2m}}$. El algoritmo 9 muestra el cálculo de la exponenciación final $\eta_T(P,Q)^M$.

Algoritmo 9: Exponenciación final del emparejamiento η_T en característica dos [5]

Entrada: $U = u_0 + u_1 s + u_2 t + u_3 st \in \mathbb{F}^*_{2^{4m}}$

Salida: $V = U^M \in \mathbb{F}^*_{2^{4m}}$ con $M = (2^{2m} + 1)(2^m - \nu 2^{\frac{m+1}{2}} + 1)$

1 $m_0 \leftarrow u_0^2$; $m_1 \leftarrow u_1^2$; $m_2 \leftarrow u_2^2$; $m_3 \leftarrow u_3^2$;
2 $T_0 \leftarrow (m_0 + m_1) + m_1 s$; $T_1 \leftarrow (m_2 + m_3) + m_3 s$;
3 $T_2 \leftarrow m_3 + m_2 s$; $T_3 \leftarrow (u_0 + u_1 s)(u_2 + u_3 s)$;
4 $T_4 \leftarrow (T_0 + T_2)$; $D \leftarrow T_3 + T_4$;
5 $D \leftarrow D^{-1}$;
6 $T_5 \leftarrow T_1 D$; $T_6 \leftarrow T_4 D$;
7 $V_0 \leftarrow T_5 + T_6$;
8 $V_1, W_1 \leftarrow T_5$;
9 **if** $\nu = -1$ **then**
10 $W_0 \leftarrow V_0$;
11 **else**
12 $W_0 \leftarrow T_6$;
13 **end**
14 $V \leftarrow V_0 + V_1 t$; $W \leftarrow W_0 + W_1 t$;
15 $V \leftarrow V^{2^m+1}$;
16 **for** $i \leftarrow 1$ **to** $\frac{m+1}{2}$ **do**
17 $W \leftarrow W^2$;
18 **end**
19 **return** VW

4.2.4. Análisis algorítmico de la exponenciación final

Del algoritmo 9 se puede observar que:

- Se tiene $U = U_0 + U_1 t$, donde $U_0 = u_0 + u_1 s$ y $U_1 = u_2 + u_3 s$. Dado que $s^2 = s + 1$ entonces:

$$\begin{aligned} U_0^2 &= (u_0^2 + u_1^2) + u_1^2 s \\ U_1^2 &= (u_2^2 + u_3^2) + u_3^2 s \\ U_1^2 s &= u_3^2 + u_2^2 s, \end{aligned}$$

  por consiguiente, obtener $T_0 = U_0^2$, $T_1 = U_1^2$ y $T_2 = U_1^2 s$ se consigue al elevar cuatro veces al cuadrado y sumar dos veces en $\mathbb{F}_{2^{2m}}$.

- La multiplicación en $\mathbb{F}_{2^{2m}}$ de la línea 3 se resuelve mediante el algoritmo Karatsuba-Ofman como se mencionó en 4.1

- Como se mencionó en 4.1 gracias a la construcción de una torre de campos finitos, la inversión en $\mathbb{F}_{2^{2m}}$ se remplaza por: una inversión, una elevación al cuadrado, tres multiplicaciones y dos sumas en $\mathbb{F}_{2^m}$.

- El siguiente paso consiste en calcular

$$\begin{aligned} V &= V_0 + V_1 t = U^{2^{2m}-1} \\ W &= W_0 + W_1 t = U^{\nu(1-2^{2m})}, \end{aligned}$$

donde V_0, V_1, W_0 y $W_1 \in \mathbb{F}_{2^{2m}}$. Se define (línea 6):

$$\begin{aligned} T_5 &= \frac{U_0^2 + U_1^2 s}{U_0^2 + U_0 U_1 + U_1^2 s} \\ T_6 &= \frac{U_1^2}{U_0^2 + U_0 U_1 + U_1^2 s}, \end{aligned}$$

es fácil ver que $U^{2^{2m}-1} = (T_5 + T_6) + T_6 t$ y $U^{1-2^{2m}} = T_5 + T_6 t$. Entonces:

$$\begin{aligned} V_0 &= T_5 + T_6 \\ W_0 &= \begin{cases} T_5 + T_6 & \text{si } \nu = -1 \\ T_6 & \text{si } \nu = 1 \end{cases} \\ V_1 &= W_1 = T_6 \end{aligned}$$

- Elevar $V = V_0 + V_1 t \in \mathbb{F}^*_{2^{4m}}$ a la potencia (2^m+1) sobre $\mathbb{F}_{2^{4m}}$ (línea 15 alg. 9) consiste en multiplicar V^{2m} por V. Esta operación resulta ser menos costosa que la multiplicación en $\mathbb{F}_{2^{4m}}$.

- En la ecuación 4.2 se puede observar que el exponente M incluye el factor $2^{(m+1)/2}$ dirigiendo al ciclo de elevaciones al cuadrado el que implica una exponenciación de $W^{(m+1)/2} \in \mathbb{F}_{2^{4m}}$. Sea $W = w_0 + w_1 s + w_2 t + w_3 st \in \mathbb{F}_{2^{4m}}$,

  calcular $U = W^2$ implica cuatro elevaciones al cuadrado y cuatro sumas en $\mathbb{F}_{2^m}$ [5]:

$$U = W^2 = (w_0^2 + w_1^2 + w_3^2) + (w_1^2 + w_2^2)s + (w_2^2 + w_3^2)t + w_3^2 st.$$

  Sin embargo, es posible librarse de la mayoría de las sumas cuando elevamos W a la potencia $\frac{m+1}{2}$. De acuerdo a la definición de s y t presentada en §4.2.3 se tiene que $s^{2^{4i}} = s$ y $t^{2^{4i}} = t$. Por lo tanto,

$$W^{2^{4i}} = w_0^{2^{4i}} + w_1^{2^{4i}} s + w_2^{2^{4i}} t + w_3^{2^{4i}} st,$$

  y esta operación requiere solamente de elevaciones al cuadrado en $\mathbb{F}_{2^m}$. Con lo anterior se puede realizar el cálculo de la línea 17 del algoritmo 9 de la siguiente

manera:

Algoritmo 10: Cálculo de $U = W^2 \in \mathbb{F}_{2^{4m}}$

Entrada: $W = w_0 + w_1 s + w_2 t + w_3 st \in \mathbb{F}_{2^{4m}}$
Salida: $U = W^2 \in \mathbb{F}_{2^{4m}}$

```
1  j ← (m + 1)/2;
2  while j mod 4 ≠ 0 do
3      W ← W^2;
4      j ← j − 1;
5  end
6  j ← j/4;
7  for (i = 1 to j) do
8      u_0 ← (w_0)^{2^4}; u_1 ← (w_1)^{2^4};
9      u_2 ← (w_2)^{2^4}; u_3 ← (w_3)^{2^4};
10 end
11 return u_0 + u_1 s + u_2 t + u_3 st
```

4.3. Implementación de exponenciación final

4.3.1. Consideraciones de diseño e implementación

En esta tesis se diseñó e implementó la exponenciación final en los campos $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{2^{313}}$, para lo cual se tomaron las siguientes consideraciones:

- se implementó el algoritmo 9 en los campos $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{2^{313}}$ considerando la aritmética descrita en §4.1 y §4.2.2.

- para $\mathbb{F}_{2^{239}}$ se emplearon los multiplicadores descritos en el capítulo 3 y para $\mathbb{F}_{2^{313}}$ fué necesario implementar el multiplicador en dicha característica siguiendo la estrategia descrita en §3.2.1.1.

- se utilizó la curva elíptica supersingular $y^2 + y = x^3 + x + b$, la cual tiene un grado de seguridad $k = 4$, donde $b = 1$.

- de acuerdo a [5] se definieron los siguientes parámetros:

	δ	ν	α	β	γ	b
$F_{2^{239}}$	1	1	0	$1 - b$	0	1
$F_{2^{313}}$	1	1	1	b	0	1

- se utilizó una representación en base polinomial para los elementos que pertenecen al campo finito, considerando a los elementos como una cadena de bits.

- se implementarón los algoritmos en la herramienta Maple para generar vectores de prueba y así poder validar los resultados obtenidos.
- se utilizó la técnica de microprogramación, generando una palabra de control de 18 bits. Para fines de programación se trabaja con una palabra de control de 20 bits dada la comodidad al usar la representación en hexadecimal.

4.3.2. Descripción de la arquitectura

La Figura 4.1 muestra el diseño para la construcción de la exponenciación final en $\mathbb{F}_{2^m}$ donde $m = 239,313$. Como se aprecia en dicha figura se requieren dos módulos principales: la unidad de control y la exponenciación final la cual a su vez esta constituida por módulos que realizan tareas específicas de acuerdo a las necesidades analizadas en la sección anterior.

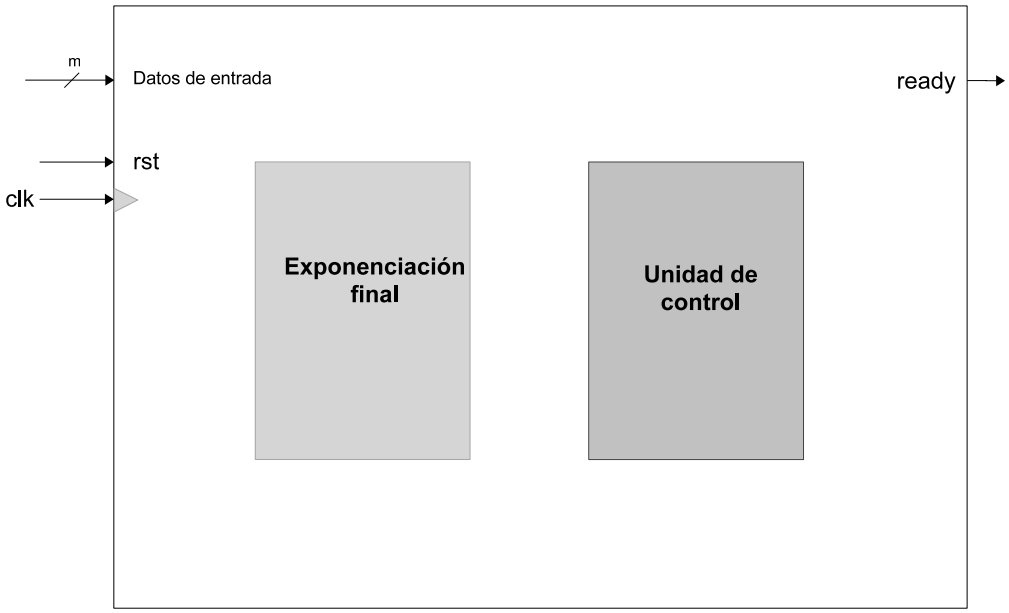

Figura 4.1: Diseño de la exponenciación final en $\mathbb{F}_{2^m}$

De manera general en la Figura 4.2 se muestra la arquitectura de hardware capaz de realizar la exponenciación final para el algoritmo de emparejamiento η_T. Se presenta la interconexión entre los módulos requeridos para evaluar la exponenciación final.

Se utilizó una memoria interna de lectura/escritura de los dispositivos de Xilinx para almancenar los operandos de entrada, los resultados parciales y el resultado final. Se analizaron todas las operaciones requeridas por el algoritmo con la finalidad de optimizar el número de registros de la memoria, se observó que existen ciertos casos donde es necesario almacenar resultados parciales por varios ciclos de reloj. Considerando lo

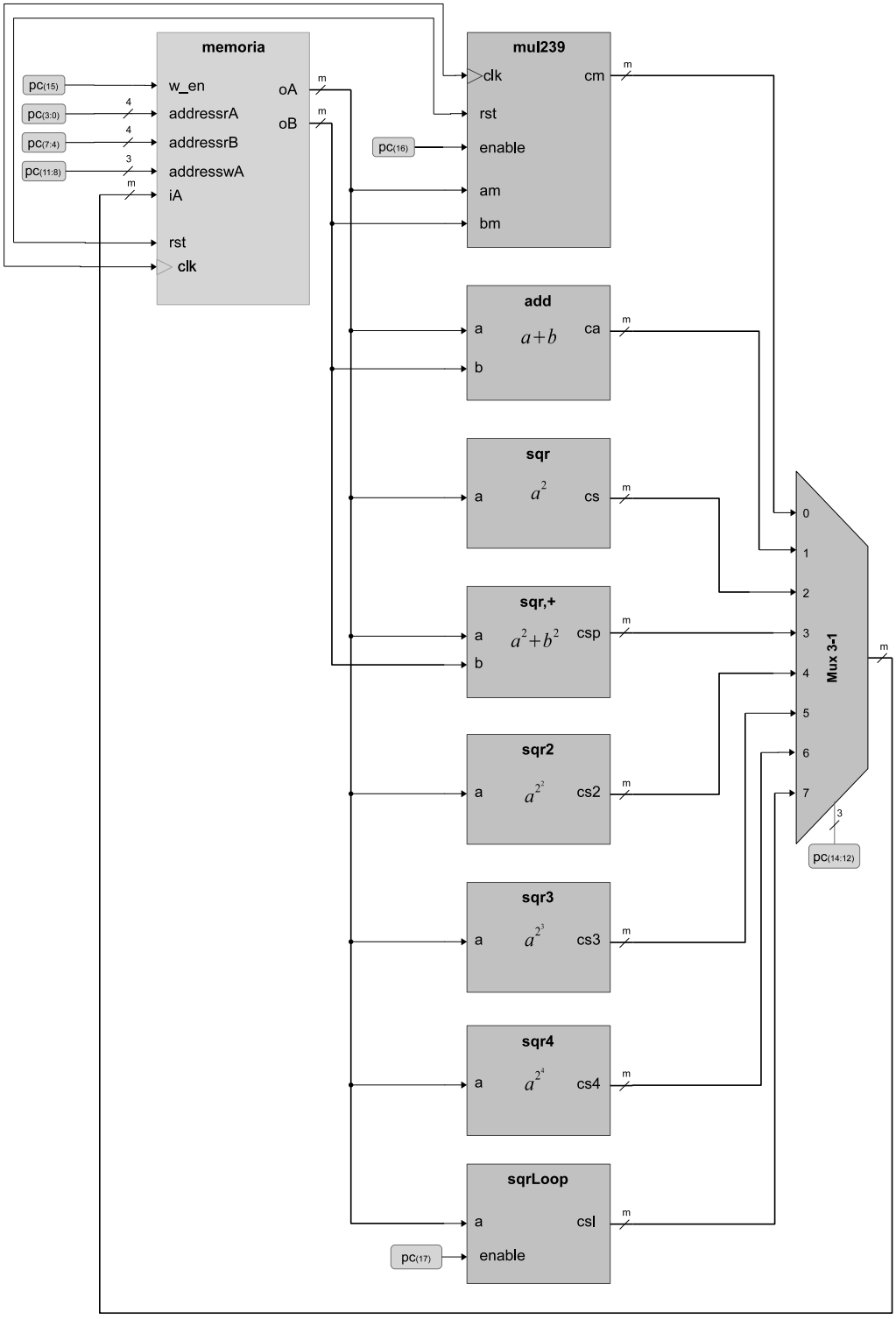

Figura 4.2: Arquitectura de la exponenciación final en $\mathbb{F}_{2^m}$

anterior, se tomó la decisión que nuestro banco de registros consistiera de once registros

de m bits para lectura y/o escritura. Los registros se nombraron siguiendo la numeración en hexadecimal del número 1 a B. Dada la construcción de nuestra memoria, es posible leer y escribir datos de manera simultánea ya que en todo momento todos y cada uno de los módulos leen los datos que estén en sus entradas y únicamente es necesario indicar la dirección del registro y el instante en que se debe de escribir.

Todos los módulos de nuestra arquitectura evaluan su operación correspondiente con los datos que tengan a la entrada, sin embargo tanto el multiplicador como el módulo dedicado al ciclo de elevaciones al cuadrado necesitan ser activados y será en ese ciclo donde lean los datos a la entrada para empezar a procesar. Se requirió una señal de control encargada de manejar al multiplexor que decide cuál resultado será almancenado.

Por lo anterior para lograr el correcto funcionamiento de la arquitectura de la exponenciación final, se necesita de una unidad de control, véase figura 4.1 la cual sirve, como su nombre lo dice, para brindar control en la trayectoría de datos entre los módulos interconectados. En la figura 4.3 se puede ver la colaboración entre la unidad de control y el módulo de exponenciación final.

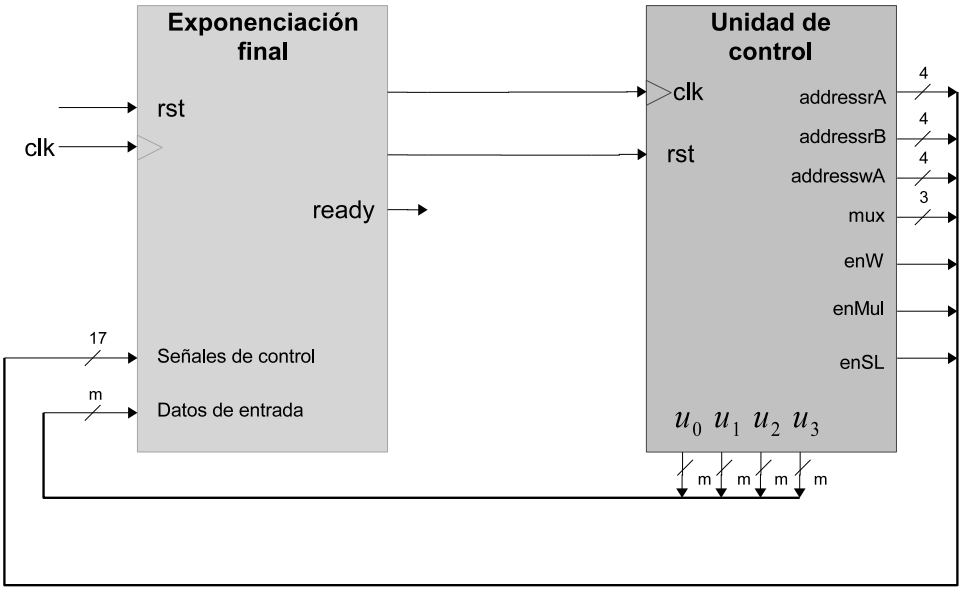

Figura 4.3: Unidad de control y trayectoría de datos para la exponenciación final en $\mathbb{F}_{2^m}$

En general, el comportamiento de nuestra arquitectura depende de palabras de control, como se mencionó, para ambas características la palabra de control es de 18 bits, pero para fines de programación se escribe como una palabra en hexadecimal de 5 dígitos. La estructura de la palabra de control se describe a continuación:

PC $_{(3:0)}$	Dirección para leer el operando A [addressrA]
PC $_{(7:4)}$	Dirección para leer el operando B [addressrB]
PC$_{(11:8)}$	Dirección para escribir el resultado de la operación [addresswA]
PC$_{(14:12)}$	Señal de control para el multiplexor [mux]
PC $_{(15)}$	Señal *W* para la memoria [enW]
PC $_{(16)}$	Señal para activar al multiplicador [enM]
PC $_{(17)}$	Señal para activar el ciclo de elevaciones al cuadrado[enSL]

El diseño que se muestra en la figura 4.1 se utilizó para la construcción de la exponenciación final en los campos $\mathbb{F}_2^{239}$ y $\mathbb{F}_2^{313}$ utilizando 371 y 429 ciclos respectivamente.

Por simplicidad de explicación en la figura 4.4 sólo se muestran las operaciones que involucra la multiplicación en $\mathbb{F}_{2^{2^m}}$ que se requiere en la línea 3 del algoritmo 9 a pesar de que en esos ciclos se estan realizando otras operaciones concurrentemente, lo anterior con el fin de ejemplificar la manera en cómo funciona nuestra arquitectura. La ecuación 4.3 desglosa los cálculos que se requieren en el campo $\mathbb{F}_{2^m}$ y en la tabla 4.1 se presenta la forma en que se construyen las palabras de control para que nuestra arquitectura realice el cómputo de la multiplicación en $\mathbb{F}_{2^{2^m}}$. También se puede notar que dada la construcción del algoritmo y ya que no existen dependencias es posible comenzar a evaluar la línea 3 del algoritmo 9 en el primer ciclo en paralelo con las operaciones de la línea 1 y de esta manera disminuir el número de ciclos requeridos para el cálculo total de la exponenciación final.

$$\begin{aligned} UV &= (u_0 + u_1 s)(v_0 + v_1 s) \\ &= u_0 v_0 + (u_0 v_1 + u_1 v_0)s + u_1 v_1 s^2 \\ &= u_0 v_0 + u_1 v_1 + (u_0 v_1 + u_1 v_0 + u_1 v_1)s \\ &= u_0 v_0 + u_1 v_1 + ((u_0 + u_1)(v_0 + v_1) + u_0 v_0)s \end{aligned} \tag{4.3}$$

4.3.3. Análisis de costos en términos de números de ciclos

La exponenciación final se implementó de acuerdo al algoritmo 9, se analizó la estructura de dicho algoritmo y se pudo notar que existen muchas dependencias entre las operaciones situación que no nos brindó la oportunidad de realizar el cálculo de dos o más operaciones de forma paralela, lo que implica que la mayoría de las operaciones deban ser evaluadas de manera secuencial.

Considerando lo anterior y al analizar las necesidades del algoritmo se diseñó una arquitectura orientada hacia la optimización del número total de ciclos requeridos para el cálculo de la exponenciación final. Por lo tanto se decidió utilizar un multiplicador rápido aunque un poco costoso el cual es capaz de realizar el cálculo de una multiplicación en $\mathbb{F}_{2^m}$ en 3 ciclos de reloj, un sumador y un módulo que calcula a^2; estos últimos requieren de un ciclo de reloj para regresar el resultado. Sin embargo al analizar el algoritmo 9 de manera más detallada se localizaron otras necesidades por lo que fue necesaria la construcción de

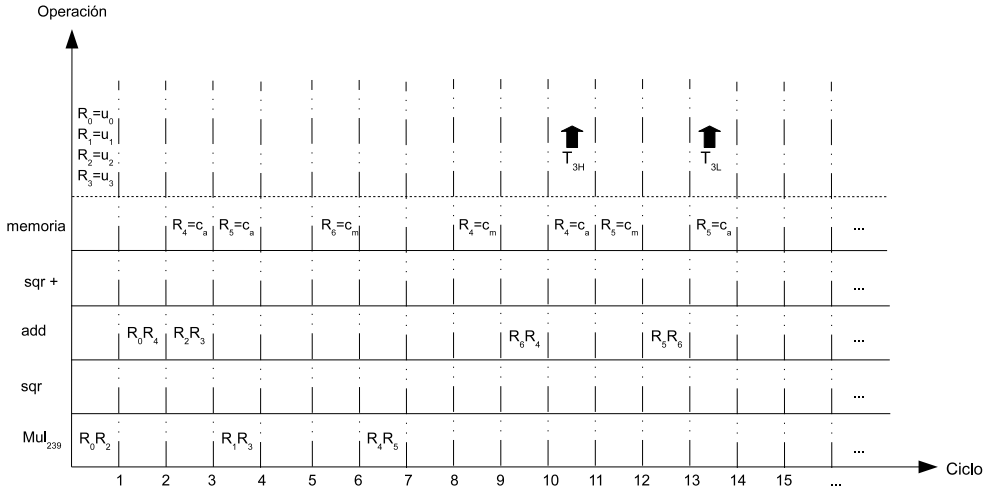

Figura 4.4: Diagrama de tiempos para evaluar una multiplicación en $\mathbb{F}_{2^{2m}}$

Tabla 4.1: Palabras de control para realizar una multiplicación en $\mathbb{F}_{2^{2m}}$

enSL	enM	enW	mux	addresswA	addressrB	addressrA	Palabra de control
0	1	0	XXX	XXXX	0010	0000	”10020”, −001
0	1	0	XXX	XXXX	0001	0000	”10010”, −002
0	1	1	001	0100	0011	0010	”19432”, −003
0	1	1	001	0101	0011	0001	”19531”, −004
0	1	0	XXX	XXXX	0011	0001	”10031”, −005
0	1	1	000	0110	0001	0000	”18610”, −006
0	1	1	011	0000	0101	0100	”1B054”, −007
0	1	0	XXX	XXXX	XXXX	0000	”10000”, −008
0	1	1	000	0100	0010	0001	”18421”, −009
0	1	1	011	0001	0100	0110	”1B146”, −010
0	1	1	001	0100	0100	0110	”19446”, −011
0	1	1	000	0101	0011	0010	”18532”, −012
0	0	1	011	0010	0110	0101	”0B265”, −013
0	0	1	001	0101	XXXX	0011	”09503”, −014
0	0	1	010	0011	XXXX	XXXX	”0A300”, −015

otros módulos más especificos. A continuación se presenta un análisis de costo en términos de números de ciclos del algoritmo 9 línea por línea.

Línea 1: requiere de cuatro elevaciones al cuadrado en el campo $\mathbb{F}_{2^m}$, dicha operación se implementó de acuerdo a la fórmula dada para los trinomios de tipo III que se presenta en §2.2.3 (en la página 16).

Línea 2: consiste en dos sumas en $\mathbb{F}_{2^m}$, véase §2.2.1 (en la página 14).

Línea 3: consideramos que: $(u_0 + u_1 s)(u_2 + u_3 s) = u_0u_2 + u_1u_3 + ((u_0 + u_1)(u_2 + u_3) + u_0u_2)s$ y de esta manera se requiere el cálculo de cuatro multiplicaciones en el campo $\mathbb{F}_{2^m}$ y tres sumas que se implementan haciendo uso de compuertas XOR.

Línea 4: como en la línea 3 del algoritmo 9 expandimos la operación teniendo así que: $T_4 = (m_0 + m_1 + m_3) + (m_1 + m_2)s$ y $D = (m_0 + m_1 + m_3 + u_0u_2 + u_1u_3) + (m_1 + m_2 + (u_0 + u_1)(u_2 + u_3) + u_0u_2)s$ lo que requiere únicamente de 7 sumas pues los productos se calcularon para la línea anterior. Utilizando nuestra arquitectura es posible ejecutar de las líneas 1 a la 4 del algoritmo 9 en 30 ciclos de reloj.

Línea 5: se trata de una inversión en el campo $\mathbb{F}_{2^{2m}}$ la cual se implementó de acuerdo al algoritmo 4 y como se mencionó en §4.2.3 esta operación se remplaza por una elevación al cuadrado, tres multiplicaciones, dos sumas y una inversión en $\mathbb{F}_{2^m}$ (alg. 4, línea 4). El cálculo del inverso multiplicativo en $\mathbb{F}_{2^m}$ se hizo utilizando el método de Itoh-Tsujii [24] con lo que el costo de la línea 5 es de trece multiplicaciones, $m + 1$ elevaciones al cuadrado y dos sumas en $\mathbb{F}_{2^m}$.

Como se observa en las tablas 2.4 y 2.5 encontrar el inverso de un elemento requiere del cálculo de potencias con exponentes grandes. Para realizar el cálculo de dichas potencias se pueden seguir distintos caminos, uno de ellos es realizar multiplicaciones consecutivas, siendo éste método muy costoso, así que considerando que en esta tesis se buscó eficiencia, al observar el comportamiento del algoritmo y dadas las necesidades se decidió implementar cuatro módulos que nos ayudaron a la construcción de la inversión sin utilizar muchos recursos en área y sin requerir de muchos ciclos de reloj.

Se decidió calcular más de una elevación al cuadrado en un solo ciclo, esto fue posible ya que dado el bajo peso Hamming de los trinomios irreducibles utilizados se pudo implementar hasta cuatro elevaciones al cuadrado consecutivas, es decir, obtener el cálculo de $a^{2^{2^4}}$ en un sólo ciclo de reloj sin penalizar la ruta crítica del diseño. Sean $a, b \in \mathbb{F}_2^m$ los módulos que se construyeron sirven para calcular: $a^{2^2}, a^{2^3}, a^{2^4}$ y $a^2 + b^2$.

De esta manera al utilizar nuestra arquitectura ahorramos 154 ciclos de reloj en $\mathbb{F}_{2^{239}}$ y 231 ciclos de reloj en $\mathbb{F}_{2^{313}}$ para obtener el resultado de la línea 5 del algoritmo 9.

Líneas 6-14: una vez que se obtuvó el inverso multiplicativo, como se muestra en el algoritmo 4, se procede con el cálculo de las multiplicaciones (alg. 4 línea 5-6) que se pueden ejecutar en paralelo con las operaciones necesarias de las líneas 6-14 del algoritmo 9, esto es, el cálculo de seis multiplicaciones y diez sumas. Nuestra arquitectura procesa lo anterior en 34 ciclos de reloj.

Línea 15: implica calcular la exponenciación en $\mathbb{F}_2^{4m}$ lo cual requiere de cinco multiplicaciones, dos elevaciones al cuadrado y nueve sumas. Sin embargo se observó que dada la estructura del algoritmo 9 no existen dependencias entre esta línea y el cálculo del ciclo de las líneas 16-18 del algoritmo 9 así que se implementaron dichas operaciones concurrentemente ahorrando ciclos de reloj.

Líneas 16-18: considerando las ideas presentadas en 4.2.4 fue posible implementar hardware específico para elevar al cuadrado un elemento del campo $\mathbb{F}_{2^{4m}}$, la arquitectura que utilizamos para implementar dicha operación se muestra en la figura 4.5. Nuestra estructura de cuatro estados nos permitió obtener el resultado de las líneas 16-18 alg.9 en apróximadamente $\lceil \frac{m+1}{4} \rceil + 3$ ciclos de reloj. Es importante mencionar que el costo de evaluar la línea 15 alg. 9 se enmascaró durante el cálculo de este ciclo de elevaciones al cuadrado.

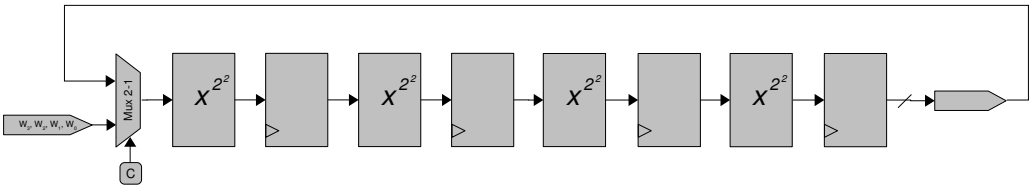

Figura 4.5: Ciclo de elevaciones al cuadrado

Línea 19: finalmente se trata de una multiplicación en $\mathbb{F}_2^{4m}$ la cual implica nueve multiplicaciones y veinte sumas en $\mathbb{F}_2^m$, se ejecutó en 56 ciclos de reloj. De modo que el número de ciclos de reloj para evaluar la exponenciación final mediante el algoritmo 9 se puede estimar así:

$$no.\ ciclos \approx 30 + \left\lceil \frac{3(m+1)}{5} \right\rceil + 40 + 34 + \left\lceil \frac{m+1}{4} \right\rceil + 3 + 56$$

4.4. Resultados

Los resultados de simulación *post place and route* en un dispositivo FPGA V2pro 50 de nuestras implementaciones se muestran en la tabla 4.2. Comparamos nuestro trabajo con otros publicados en la literatura abierta. Debido a que la mayoría de las publicaciones incluyen únicamente los datos para el cálculo del emparejamiento y no detallan sus resultados de la exponenciación final nos fue difícil poder compararlos con más trabajos, sin embargo, incluimos el trabajo de Grabher *et al.* [20] que si bien, es en $\mathbb{F}_{3^{97}}$ pero es comparable pues la seguridad que brinda es similar a la que proporciona nuestra implementación en $\mathbb{F}_{2^{239}}$.

Finalmente incluimos el trabajo de Beuchat *et al.* [6], el cual, además de ser el más reciente, obtuvo el premio al mejor artículo; analizando la tabla es posible ver que su trabajo es ventajoso contra el nuestro ya que duplican la frecuencia de reloj pero ocupan casi el doble de ciclos que nosotros lo cual les sirvió también para reducir el área; en general al observar las diferentes métricas consideramos que nuestro diseño es competitivo.

Tabla 4.2: Resultados de la exponenciación final en $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{2^{313}}$

Exponenciación Final	Ciclos	Área $[slices]$	Frecuencia $[MHz]$	Retardo $[\mu s]$	Eficiencia	Throughput [Mbits/seg]
$F_{2^{239}}$ 1e	371	7,026	83	4.45	32	54
$F_{2^{313}}$ 1e	429	10,583	84	5.11	19	61
Beuchat *et al.* $F_{2^{239}}$ [6]	688	3,039	167	4.13	79	58
Grabher *et al.* $F_{3^{97}}$ [20]	4,941	–	150	32.90	–	3

Capítulo 5
Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones con respecto a este trabajo de tesis en base a conocimientos y resultados obtenidos así como se menciona acerca del trabajo a futuro.

5.1. Sumario de actividades

En esta tesis se propusieron diseños novedosos para la construcción de multiplicadores en característica dos y característica tres, su construcción se basa en arquitecturas construidas mediante una combinación de variantes recientemente propuestas del ya conocido algoritmo de multiplicación Karatsuba-Ofman. Considerando el objetivo de obtener un multiplicador muy rápido sin incrementar la ruta crítica del algoritmo, propusimos una arquitectura segmentada que es muy útil para aplicaciones donde se requiera el cálculo de varias multiplicaciones independientes tales como el caso de estudio del emparejamiento.

Se obtuvo el multiplicador más eficiente que cualquier otro reportado en la literatura abierta en ambas características, específicamente en los campos finitos $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{3^{97}}$. En característica tres presentamos un multiplicador capaz de calcular más de una multiplicación en un sólo ciclo de reloj.

Además como aplicación a nuestros multiplicadores se diseñó e implementó una arquitectura rápida capaz de calcular la exponenciación final del emparejamiento η_T en característica dos, especificamente para los campos finitos $\mathbb{F}_{2^{239}}$ y $\mathbb{F}_{2^{313}}$.

De acuerdo a los resultados obtenidos de la simulación *post-place and route* en distintos dispositivos FPGA´s los diseños de nuestros multiplicadores mejoran tanto el tiempo de cálculo como el área que utilizan, siendo así diseños más eficientes comparados con propuestas publicadas en la literatura abierta en los campos 2^{239} y 3^{97}. En cuanto a la implementación de la exponenciación final en los campos 2^{239} y 2^{313} obtuvimos resultados muy competitivos.

5.2. Conclusiones

Como se puede ver en la tabla de resultados en §3.2.2 (en la página 42), la implementación de nuestros diseños de multiplicadores utilizando la técnica de Karatsuba-Ofman libre de traslape propuesta en [17] no mejora los resultados de la versión clásica del algoritmo Karatsuba-Ofman, esto se debe a que dada la organización de los bloques lógicos configurables (CLBs) de los dispositivos utilizados (véase Apéndice A), no se permite que exista el ahorro en compuertas XOR que ellos sugieren dando como resultado eficiencias similares a la implementación del algoritmo clásico de Karatsuba-Ofman.

En cuanto a la aplicación de nuestros multiplicadores en el cómputo de la exponenciación final se puede ver que a pesar de la dependencia entre la mayoría de las multiplicaciones de la exponenciación final, nuestros diseños consiguen una eficiencia competitiva, algunas de las ideas que propusimos en este capítulo fueron consideradas por Beuchat *et al.* en [6].

5.3. Trabajo futuro

A pesar de que nuestros diseños presentan una mayor eficiencia contra cualquier otro trabajo publicado en la literatura abierta, mientras se ahondaba más en el estudio de esta tesis se encontraban mejoras posibles en nuestros diseños, sin embargo por factores de tiempo, no se pudieron realizar. De tal manera que los resultados presentados en esta tesis pueden ser mejorados aprovechando técnicas aritméticas en la implementación de la exponenciación final tratando de ahorrar ciclos de reloj o en la construcción de los multiplicadores con diseños que sirvan para reducir aún más la trayectoría crítica.

Apéndice A
Dispositivos FPGAs y el lenguaje de descripción de hardware VHDL

A.1. FPGA

Los dispositivos FPGA (del inglés Field Programmable Gate Array) se introdujeron por Xilinx a mediados de los años ochentas. A grandes rasgos las FPGA´s se construyen utilizando compuertas lógicas. La arquitectura de las FPGA´s consiste en una matriz de bloques lógicos configurables (CLB por sus siglas en inglés), donde la lógica y la conexión pueden ser programadas.

El diseño de los bloques CLB varia de fabricante en fabricante incluso de dispositivo en dispositivo. Un CLB puede ser tan simple como una tabla de consulta (LUT por sus siglas en inglés) de cuatro entradas o tan compleja como una unidad aritmética lógica (ALU) de cuatro entradas o como una LUT de seis entradas [42].

Se acostumbra definir la granularidad de la lógica reconfigurable como el tamaño de la unidad funcional más pequeña que puede ser direccionada por las herramientas de programación. Aquellas arquitecturas que tengan la granularidad más baja tienden a ser muy útiles para manipulación de datos a nivel de bits y, en general, para circuitos combinacionales. Por otro lado, los bloques con granularidad alta son convenientes para los niveles más altos de la manipulación de datos, por ejemplo, para desarrollar circuitos a nivel de transferencia de registros.

La interconexión en un FPGA juega un papel importante en un dispositivo FPGA, debido a la necesidad de una rápida y eficiente línea de comunicación entre los diferentes bloques lógicos los cuales estan organizados en filas y columnas.

Existen muy pocos fabricantes de FPGA´s comerciales y usualmente cada uno de ellos ha desarrollado una o más familias de dispositivos. La Tabla A.1 muestra algunos de los fabricantes más populares.

En esta tesis se utilizaron los dispositivos FPGAs de Xilinx, principalmente las familias

Tabla A.1: Fabricantes de FPGA y sus dispositivos

Fabricante	Familia FPGA	Características
Xilinx	Virtex-5, Virtex-4 Virtex II, Virtex II-Pro	Líder del mercado FPGA Tecnología 65*nm*
Altera	Stratix, Stratix II, Cyclone	Tecnología 90*nm*
Lattice	LatticeXP	Primer FPGA no volátil
Actel	Fusion, M7Fusion	Primer FPGA de señales mixtas
Quick Logic	Eclipse II	FPGA programable una sola vez
Atmel	AT40KAL	Reconfigurable de granularidad fina

Virtex-5 y Virtex II Pro. La arquitectura para estas familias consiste en cinco elementos funcionales fundamentales los cuales son:

- Bloques lógicos configurables (CLB por sus siglas en inglés)
- Bloques de Entrada /Salida (IOBs por sus siglas en inglés)
- Bloque de memoria de acceso aleatorio (RAM por sus siglas en inglés)
- Multiplicadores dedicados
- Administrador de reloj digital (DCMs)

Los CLBs son los recursos de hardware más abundantes e importantes de un FPGA. Típicamente se utilizan tanto para diseños lógicos combinacionales como en diseños lógicos síncronos. Cada CLB esta compuesto por cuatro capas (slices) agrupadas por pares y cada uno de estos pares está en una columna con un acarreo independiente. Todas las capas tienen los mismos elementos que son:

- dos LUT
- dos flip flops tipo D
- multiplexores
- circuitos lógicos para el manejo del acarreo
- compuertas de aritmética lógica

Ambos pares de *slices* utilizan dichos elementos para proveer funciones lógicas, aritméticas y como memoria de sólo lectura. Un CLB puede ser configurado para trabajar en dos modos: lógico y memoria. En el modo lógico cada LUT pertenece a un CLB y un bit de registro. En los dispositivos Xilinx las LUTs pueden reprogramarse para cualquier función lógica de cuatro entradas una sálida. En modo memoria los bloques LUT se comportan como dos pequeños bloques de memoria.

A.2. VHDL

En este apendice se presenta una breve reseña de la plataforma de desarrollo incluyendo la descripción del dispositivo utilizado, las herramientas y el lenguaje.

A.2.1. Plataforma de desarrollo

Las implementaciones de ésta tesis se realizaron con ayuda de las herramientas de diseño que provee Xilinx:

ISE (Integrated Software Enviroment) Entorno integrado de desarrollo que permite recorrer fácilmente las diferentes fases del proceso de desarrollo de un circuito lógico, desde el diseño hasta la implantación sobre una arquitectura reconfigurable. ISE se encarga de llamar de manera automática a las diferentes utilidades y herramientas suministradas dependiendo de la etapa de diseño en la que nos encontremos. ISE se autodenomina Navegador de Proyecto (Project Navigator) y está orientado a facilitar el proceso de desarrollo. Las versiones utilizadas son: ISE Project Navigator 8.1 y 9.2 para el sistema operativo Microsoft Windows disponible en:

http://www.xilinx.com/ise/logic_design_prod/foundation.htm

Modelsim Herramienta que provee un ambiente de simulación completo que permite verificar los modelos funcionales y de tiempo de cualquier diseño así como el código fuente. Está optimizado para usarse con todas las configuraciones de Xilinx ISE. La versión utilizada es: Model*Sim* SE/PE/LE 6.2c

VHDL Lenguaje de descripción de hardware.

Xilinx es la industria más completa en soluciones para diseño de lógica programable que brinda: rendimiento óptimo, poder de administración, reducción de costo y productividad. ISE Foundation ofrece soporte para todos los principales FPGAs.

El dispositivo que se utilizó para los diseños de ésta tesis corresponde a la familia Virtex 5 (XC5VLX330) el cual satisface las necesidades de desarrollo como son área y velocidad. Los resultados que se presentaran se obtienen a través de la herramienta de síntesis incluida en el navegador de proyecto de Xilinx.

Dispositivo XC5VLX330			
Slices	I/O	CLB	BRAM
51,840	1,200	3,420	10,368

A.2.2. Lenguaje de descripción de hardware VHDL

VHDL es un lenguaje para describir sistemas electrónicos digitales. Surgió a partir del programa de Circuitos Integrados de muy Alta Velocidad (VHSIC por sus siglas en inglés) del gobierno de los Estados Unidos, el cual inició en 1980. Durante dicho programa, se hizo notoria la necesidad de contar con un lenguaje estándar para describir la estructura y la función de los circuitos integrados (IC por siglas en inglés). Esto derivó en el desarrollo del Lenguaje de Descripción del Hardware VHSIC (VHDL por sus siglas en inglés), el cual fue posteriormente adoptado por la IEEE como un estándar [39].

El lenguaje VHDL está diseñado para satisfacer determinadas necesidades en el proceso de diseño [39]:

1. Permite la descripción de la estructura del diseño, lo que representa la manera en que está dividida en sub-diseños, y cómo esos sub-diseños son interconectados.

2. Permite que las funciones sean especificadas mediante el uso de estructuras de lenguajes de programación que son familiares a los desarrolladores.

3. Como resultado, es posible realizar la simulación del diseño antes de ser fabricado, por lo que los diseñadores son capaces de comparar las alternativas y probar que sean correctos sin el retraso y el costo del desarrollo de prototipos de hardware.

VHDL divide las entidades (componentes, circuitos o sistemas) en una parte visible o externa (nombre de la entidad y conexiones) y una parte oculta o interna (algoritmo de la entidad e implementación). Después de definir la interfaz externa, otras entidades pueden usarla cuando haya sido completamente desarrollada. Este concepto de vista externa e interna es central para una perspectiva de diseño de sistemas en VHDL. Una entidad es definida, en relación a otras, por sus conexiones y comportamiento. Se pueden explotar implementaciones externas (arquitecturas) de una de ellas sin cambiar el resto del diseño. Después de definir una entidad para un diseño, se puede reutilizar en otros diseños tanto como sea necesario.

Una entidad o diseño VHDL tiene uno o más puertos de entrada, salida o entrada/salida, los cuales son conectados a sistemas vecinos. Está formada de procesos y componentes interconectados, todos operando concurrentemente. Cada entidad es definida por una arquitectura particular, que se compone de construcciones VHDL tal como operaciones aritméticas, asignaciones de señal, o declaraciones de instanciación de componentes.

En VHDL, los circuitos de modelo secuencial en procesos independientes (síncronos), usan flip-flops y latches, y los circuitos combinacionales (asíncronos) utilizan típicamente compuertas lógicas. Los procesos pueden definir y llamar (instanciar) subprogramas (diseños secundarios). Los procesos se comunican con cada uno de los otros procesos mediante señales.

Una señal tiene una fuente (manejador), uno o más destinos (receptores), y un tipo definido por el usuario.

Las construcciones del lenguaje VHDL están divididas en tres categorías de acuerdo a su nivel de abstracción:

comportamiento: Es posible describir un circuito electrónico, generalmente digital, simplemente describiendo el comportamiento funcional o algorítmico del diseño, expresado en un proceso secuencial VHDL.

flujo de datos: Se ven a los datos como un flujo a través del diseño, de la entrada a la salida. Una operación es definida en términos de una colección de transformación de datos, expresado como una sentencia concurrente.

estructural: La visión más cercana al hardware; un modelo donde los componentes de un diseño se enumeran y se interconectan. Está expresada por instanciación de componentes.

Todos los diseños que se presentan en esta tesis utilizaron una construcción de tipo estructural.

A.2.3. Diseño en FPGA

Para realizar las implementaciones se siguieron los siguiente pasos durante el diseño:

1. Análisis de los algoritmos a implementar: se determinaron los bloques básicos a implementar, con la finalidad de obtener los resultados esperados.

2. Optimización de la arquitectura: se analizó la secuencia del proceso, con la finalidad de encontrar posibles mejoras para el desempeño.

3. Implementación de los bloques básicos en VHDL: una vez identificados los componentes básicos que constituyen el sistema completo, se implementaron partiendo desde los más sencillos hasta agruparlos para formar los más complejos.

4. Simulación de cada uno de los componentes implementados: esta simulación se realizó con la herramienta Model*Sim*, con la que fue posible analizar a detalle el comportamiento y realizar pruebas que permitieron validar el funcionamiento.

5. Validación del diseño mediante la síntesis: una vez que se conto con la simulación funcional se realizo la síntesis para asegurar que no existían errores.

Bibliografía

[1] O. Ahmadi and F. Rodríguez-Henríquez. Low complexity cubing and cube root computation. Cryptology ePrint Archive, Report 2009/070, 2009. `http://eprint.iacr.org/`.

[2] P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In M. Yung, editor, *Advances in Cryptology – CRYPTO 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–368. Springer, 2002.

[3] G. Bertoni, J. Guajardo, S. Kumar, E. Kumar, G. Orlando, C. Paar, T. Wollinger, and G. Orlandogdcs Corn. Efficient $gf(p^m)$ arithmetic architectures for cryptographic applications. In *In Topics in Cryptology - CT RSA 2003*, pages 158–175. SpringerVerlag, 2003.

[4] J. L. Beuchat, N. Brisebarre, J. Detrey, and E. Okamoto. Arithmetic operators for pairing-based cryptography. In P. Paillier and I. Verbauwhede, editors, *Proceedings of CHES 2007*, number 4727 in Lecture Notes in Computer Science, pages 239–255. Springer, 2007.

[5] J.L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez. A comparison between hardware accelerators for the modified Tate pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. Cryptology ePrint Archive, Report 2008/115, 2008. `http://eprint.iacr.org/`.

[6] J.L. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-Henríquez. Hardware accelerator for the Tate pairing in characteristic three based on Karatsuba-Ofman multipliers. In C. Clavier and K. Gaj, editors, *Cryptographic Hardware and Embedded Systems – CHES 2009*, number 5747 in Lecture Notes in Computer Science, pages 225–239. Springer, 2009.

[7] J.L. Beuchat, H. Doi, K. Fujita, A. Inomata, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, T. Takagi, A. Vithanage, and H. Yamamoto. FPGA and ASIC implementations of the η_T pairing in characteristic three. Cryptology ePrint Archive, Report 2008/280, 2008.

[8] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. In J. Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 213–229. Springer, 2001.

[9] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *Advances in Cryptology – ASIACRYPT 2001*, number 2248 in Lecture Notes in Computer Science, pages 514–532. Springer, 2001.

[10] N. S. Chang, C. H. Kim, Y. Park, and J. Lim. A non-redundant and efficient architecture for karatsuba-ofman algorithm. In J. Zhou, J. Lopez, R. H. Deng, and F. Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 288–299. Springer, 2005.

[11] N. Cortez-Duarte, F. Rodríguez-Henríquez, J. L. Beuchat, and E. Okamoto. A pipelined Karatsuba-Ofman multiplier over GF(3^{97}) amenable for pairing computation. Cryptology ePrint Archive, Report 2008/127, 2008.

[12] N. Cruz-Cortés, F. Rodríguez-Henríquez, and C. A. Coello-Coello. On the optimal computaion of finite field exponentiation. In C. Lemaître, C. A. Reyes, and J. A. González, editors, *IBERAMIA*, volume 3315 of *Lecture Notes in Computer Science*, pages 747–756. Springer, 2004.

[13] A. J. Menezes D. R. Hankerson, S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[14] I. Duursma and H.S. Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In C.S. Laih, editor, *Advances in Cryptology – ASIACRYPT 2003*, number 2894 in Lecture Notes in Computer Science, pages 111–123. Springer, 2003.

[15] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *CRYPTO 84*, pages 10–18, 1985.

[16] N. Estibals. Cryptographie à base de couplages et multiplieurs parallèles en caractéristiques 2 et 3, 2008. Rapport de stage de Master 1.

[17] H. Fan, J. Sun, M. Gu, and K. Y. Lam. Overlap-free Karatsuba-Ofman polynomial multiplication algorithm. 2007. `http://eprint.iacr.org/`.

[18] G. Frey and H.-G. Rück. A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. *Mathematics of Computation*, 62(206):865–874, April 1994.

[19] S.D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate pairing. In C. Fieker and D.R. Kohel, editors, *Algorithmic Number Theory – ANTS V*, number 2369 in Lecture Notes in Computer Science, pages 324–337. Springer, 2002.

[20] P. Grabher and D. Page. Hardware acceleration of the Tate pairing in characteristic three. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 398–411. Springer, 2005.

[21] D. Hankerson, A. J. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, New Jersey, USA, 2003.

[22] G. Hanrot and P. Zimmermann. A long note on mulders' short product. *J. Symb. Comput.*, 37(3):391–401, 2004.

[23] K. Harrison, D. Page, and N. Smart. Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems. *London Mathematical Society J. Comput. Math.*, 5(-):181–193, November 2002.

[24] T. Itoh and S. Tsujii. A Fast Algorithm for Computing Multiplicative Inverses in Finite Fields Using Normal Basis. pages 1637–1645, 1989.

[25] A. Joux. A one round protocol for tripartite Diffie-Hellman. In W. Bosma, editor, *Algorithmic Number Theory – ANTS IV*, number 1838 in Lecture Notes in Computer Science, pages 385–394. Springer, 2000.

[26] A. Karatsuba and Y. Ofman. Multiplication of multidigits numbers on automata. volume 7, pages 595–596, January 1963.

[27] T. Kerins, W.P. Marnane, E.M. Popovici, and P.S.L.M. Barreto. Efficient hardware for the Tate pairing calculation in characteristic three. In J.R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 412–426. Springer, 2005.

[28] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[29] N. Koblitz. Elliptic curve in cryptography. *American Mathematical Society J. Comput. Math.*, 48(177):207–209, January 1987.

[30] S. Leilei and P. Keshab. Low-energy digit-serial/parallel finite field multipliers. *J. VLSI Signal Process. Syst.*, 19(2):149–166, 1998.

[31] A. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curves logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, September 1993.

[32] A. J. Menezes. An introduction to pairing-based cryptography. 2005. Notes from lectures given in Santander, Spain.

[33] A. J. Menezes, S. A. Vanstone, and P. C. Van-Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, October 1996.

[34] V. S. Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer science; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[35] V.S. Miller. Short programs for functions on curves. Available at `http://crypto.stanford.edu/miller`, 1986.

[36] V.S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[37] S. Mitsunari, R. Sakai, and M. Kasahara. A new traitor tracing. *IEICE Trans. Fundamentals*, E85-A(2):481–484, Feb 2002.

[38] D. Page and N. Smart. Hardware implementation of finite fields of characteristic three. In *Cryptographic Hardware and Embedded Systems (CHES)*, pages 529–539. Springer-Verlag, 2003.

[39] J. A. Peter. In *The VHDL Cookbook*, 1990.

[40] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

[41] F. Rodríguez-Henríquez, G. Morales-Luna, and J. López-Hernández. Low complexity bit-parallel square root computation. Cryptology ePrint Archive, Report 2006/133, 2006. `http://eprint.iacr.org/`.

[42] F. Rodríguez-Henríquez, A. Díaz-Pérez N.A. Saquib, and C. Kaya-Koc. *Cryptographic Algorithms on Reconfigurable Hardware*. Springer, 1st edition, November 2006. ISBN: 0387338837.

[43] R. Ronan, C. Murphy, T. Kerins, C. Ó hÉigeartaigh, and P.S.L.M. Barreto. A flexible processor for the characteristic 3 η_T pairing. *Int. J. High Performance Systems Architecture*, 1(2):79–88, 2007.

[44] R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, and T. Kerins. FPGA acceleration of the Tate pairing in characteristic 2. In *Proceedings of the IEEE International Conference on Field Programmable Technology – FPT 2006*, pages 213–220. IEEE, 2006.

[45] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *2000 Symposium on Cryptography and Information Security (SCIS2000), Okinawa, Japan*, pages 26–28, January 2000.

[46] B. Schneier. *Applied Cryptography, Protocols, Algorithms, and Source Code in C*. Wiley, 2nd edition, 1996.

[47] B. Schneier. *Cryptography and Network Security, Principles and Practice*. Prentice Hall, 3rd edition, 2003.

[48] A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO'84 of LNCS*, volume 196, pages 47–53, Berlin, Germany, 1985. Springer-Verlag.

[49] J. Shokrollahi, E. Gorla, and C. Puttmann. Efficient fpga-based multipliers for $\mathbb{F}_{3^97}$ and $\mathbb{F}_{3^{(6*97)}}$. In K. Bertels, W. A. Najjar, A. J. van Genderen, and S. Vassiliadis, editors, *FPL*, pages 339–344. IEEE, 2007.

[50] C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. In *Proceedings of the IEEE International Conference on Field Programmable Technology – FPT 2006*, pages 173–180. IEEE, 2006.

[51] Andrew S. Tanenbaum. *Redes de Computadoras*. Prentice Hall, 2003.

[52] L. C. Washington. Elliptic curves: Number theory and cryptography. *CRC, Secaucus*, 2003.